Data-Flow Expression Evaluator for VLSI Implementation

by

Faaez Mohammed Ghaffar

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

ELECTRICAL ENGINEERING

June, 1987

Order Number 1355749

Data-Flow Expression Evaluator for VLSI implementation

Ghaffar, Faaez Mohammed, M.S.

King Fahd University of Petroleum and Minerals (Saudi Arabia), 1987

U·M·I

300 N. Zeeb Rd.
Ann Arbor, MI 48106

DATA-FLOW EXPRESSION EVALUATOR FOR VLSI IMPLEMENTATION

BY

FAAEZ MOHAMMED GHAFFAR

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

ELECTRICAL ENGINEERING

JUNE 1987

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS

DHAHRAN, SAUDIA ARABIA

This thesis, written by MR. FAAEZ MOHAMMED GHAFFAR under the direction of his Thesis Committee, and approved by all its members, has been presented to and accepted by the Dean, College of Graduate Studies, in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE IN ELECTRICAL ENGINEERING.

Dean, College of Graduate Studies

Date: 4/10/87

Department Chairman 4/10/87

THESIS COMMITTEE

Dr. Gerhard F. Beckhoff, Chairman

Dr. Mushfiqur Rahman, Member

Dr. Zararullah Zafar, Member

DEDICATION

This thesis is dedicated to my family

ACKNOWLEDGMENTS

Acknowledgment is due to the King Fahd University of Petroleum and Minerals for support of this research.

I wish to express my appreciation to Professor G. F. Bekhoff , who served as my major advisor. I also wish to thank the other members of my Thesis Committee, Dr. Z . Zafar and Dr. Mushfiqur-Rehman.

I am thankful to Dr. F. M. Zedan for his guideance and help throughout my academic years at KFUPM. I am also grateful to Dr. Talal. Halawani and Dr. M. Dawood for their guideance and help in my undergraduate years at KFUPM.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

This work involves the design and development of a numeric expression evaluating computing system which can be added as a special purpose slave processor to a host computer. The Data-Flow Expression Evaluator (DFEE) would increase the performance of the host computer in terms of computation speed-up. The expression evaluation system is based on data-flow computing architecture. The architecture is further developed into a highly parallel circular pipelined computing architecture. This architecture is mapped into a Very Large Scale Integrated Circuit (VLSI) design which would be suitable for single chip VLSI implementation.

خلاصـــــــــه

يتضمن هـــذا العمـــل تصميم وتطوير التعبيـــر الرقمى لتقويـــم نظام الحســـاب الذى يمكـــن اضافتـــه كغرض خاص معالـــج تابـــع للحاســـب الالـــى المضيـــف . المقوم التعبيـــرى لتدفـــق المعلومات سيزيـــد فاعليـــة الحاسب الالى المضيـــف من ناحيـــة اســـراع الحســـاب .

يعتمـــد نظام التقويم التعبيرى على بنـــاء تدفـق المعلومات الحسابـــى البنـــاء اضافـــة الى ذلك مطـــور الى بنـــاء حسابـــى عالى متوازى دائـــرى خط مواسيرى .

هـــذا البناء مخطط الى تصميـــم مقايس دائره متكامـــل كبير جـــدا الذى يكون مناسبا لجذاذة (شـــب) وحيـــده لتنفيـــذ مقياس متكامـــل كبير جـــدا.

CHAPTER 1

INTRODUCTION TO DATA-FLOW CONCEPTS

1.1 INTRODUCTION

In the world of computing, there has always been a demand for time efficient and high speed computing. Presently, this demand has been increased because more sophisticated applications of today require a computing system with enormous memory and computing power in terms of time and bandwidth (bandwidth: the rate of instructions executed per unit time). These applications are listed below in increasing order of complexity [1]:

i) Date Processing,

ii) Information Processing,

iii) Knowledge Processing,

iv) Intelligence Processing,

One way of dealing with this problem is to employ high speed logic using faster technologies. However, there is a limit to this, since the constraint on high speed computing is physical in nature,i.e., inherent delays of the devices. In addition, the faster technologies require high power consumption due to the inverse relation of speed and power in high speed devices[10].

This demand has led researchers towards parallel processing. Parallel processing is an efficient form of computing which emphasizes the exploitation of concurrent events in the computing process. Concurrency implies parallelism, simultaneity and pipelining. Parallel events may occur in multiple resources during the same time interval; simultaneous events may occur at the same time instant and pipelined events may occur in overlapped time spans [1].

In parallel processing, faster execution is attained by parallel computing architectures in addition to high speed logic employed in the system.

For more than thirty years the principles of computer architecture design have largely remained static, based on the von Neumann organization [15]. These von Neumann principles include :

i) A single computing element incorporating processor, communication and memory.

ii) Linear organization of fixed size , shared memory cells.

iii) One level address space of cells.

iv) Low-level machine language.

v) Sequential, centralized control of computation.

As the need for more efficient computing came up, researchers revisited this organization and included into it more processing elements and memory modules. The reason for doing so was simple: to execute as much computation as possible in the same time interval using these parallel resources.

The sequential, centralized control of computation was augmented

by inclusion of such parallel control operators as FORK and JOIN. However, the control flow still remained sequential unless FORK and JOIN were embedded into the control flow at the program level. Since parallel and concurrent operations were now described at the program level, this implied that some constraint needed to be put on the data which would utilize this parallelism. If this was not so, a redefined control flow at the program level was required.

Highly parallel computers such as the Illiac IV and CDC Star were based on the same concept. These computers achieved there processing speed by imposing constraints on the structure of the data being processed. Both of these machines are organized to perform very well for data represented as vectors. In both these highly parallel machines, the programmer is forced to use unusual and intricate data representations, if highly parallel execution is to be achieved.

The other approaches to parallel processing basically involved multi processor memory configurations. In practical computer systems these approaches have not proved to be successful. The reason for this is, no matter how decentralized the processor memory is made, there is always a requirement to share the data computed from one processor memory module to another such processor memory module. Thus as system size increases, these systems grow in complexity of the processor memory communication switches[14].

The processor memory module architecture presented above is decentralized. Each module would execute the sub-tasks independent of each other's operation. However, there is communication between

the processor memory modules since they may need to share the data depending on the sub-tasks assigned to them. If the sub-tasks are entirely independent of each other, then each processor memory module would carry out the sub-task without the need of communicating with the rest of the processor memory module. After the processor memory module finishes this sub-task, it can take on another independent sub-task and carry out the computation. Therefore, this process of independent execution of independent sub-tasks can go on until a complete task is finished. The advantage of this approach would be maximum utilization of the processor-memory module without idle time (idle time: the time interval during which the processor memory module is idle)[2].

The above description leads to the question of whether any task can be sub-divided into sub-tasks such that each sub-task can be executed independently on a processor memory module. More precisely, does there exist an execution control [3] which enables the computational task or algorithm to be partitionned into independent sub-tasks and to execute them concurrently ?

1.2 THE DATA-FLOW CONCEPT

Current interest in data-driven or data-flow computing has shown that it can be done so. Data-flow or data-driven mechanisms of execution control have been introduced in an attempt to easily sequence the operations in a fully distributed system and to afford easy programmability in multiprocessing systems [4].

The data-flow concept introduces a fundamentally different way of instruction execution compared to the traditional von Neumann architecture. Data-flow or data-driven execution control naturally exploits the inherent parallelism of a computing task. Instead of a central controller (the program counter) as would exist in a von Neumann style machine, data carry with them their own control in the form of pointers that contain the address of the instructions which would be using the attached values. As soon as the data is received, the instruction is ready for execution. The result of the executed instruction is then given to the instruction requiring the result from the previous executed instruction.

The availability of data-values prescribe the execution of instructions. The availability of data-values in turn represents the task to be computed. Thus the task is divided automatically into sub-tasks to be executed concurrently in the multiprocessing system.

The sub-tasks which remain lower down in the hierarchy of execution are awaiting the results of the previous sub-tasks which are being executed in the processor memory module concurrently.

1.3 DATA-FLOW GRAPH

The task or algorithm description in a data-flow environment is known as a data-flow graph. Consider the following computation :

a:= (b x d) + (c x d)

where a, b, c and d represent data-values. This computation has been mapped into the corresponding data-flow graph in Figure 1.1.

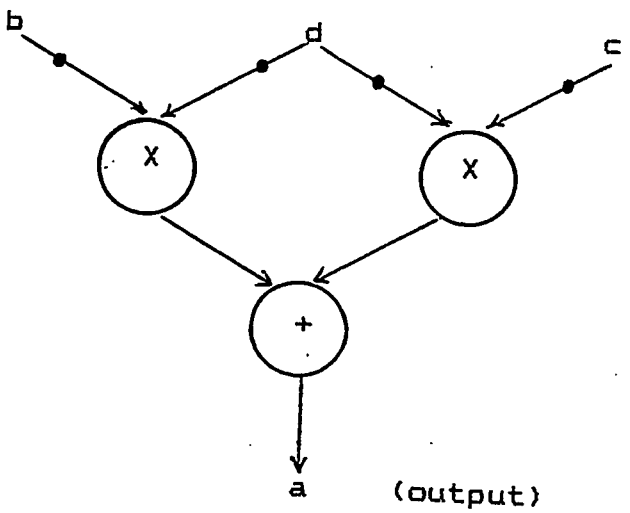


FIGURE 1.1: DATA-FLOW GRAPH OF a:=(b x d) + (c x d)

Data-flow graphs are directed graphs illustrating the flow of data between instructions. The set of instructions compromises the computation to be performed in the data-flow environment. In the data-flow graph representation, each instruction consists of an operator, two operands (one operand in the case of a unary operator like square-root). These operands are either literal operands (constants) or unknown operands defined by their previous predecessor instructions. The black dots shown in Figure 1.1 are data-tokens.

An instruction is enabled for execution when all the input data-tokens are available. That is , when all unknowns have been replaced by partial results made available by other predecessor instructions. The literals are constant values at the initial level. Since they are always available, the instructions in the initial level are always enabled. The resulting data-token is carried on to the next level, the second level. This is shown in Figure 1.1.

The execution levels correspond to the instruction levels,.i.e, level i will always be executed before level i+1 lower down in the directed graph. The computations belonging to the same level are independent of each other and therefore can be executed concurrently.

1.4 DATA-FLOW MODEL: PROPERTIES AND PRINCIPLES

The data-flow systems consists of the following properties:

i)Asynchronomy

The data-flow concepts mean asynchronomy because each computation task can be divided into sub-tasks which are independent of each other. Therefore, the sub-tasks can be executed asynchronously on parallel resources. Due to this, a high degree of implicit parallelism is available in data-flow systems [1].

ii)No side effects

Because there is no use of shared memory cells, the data-flow system of computation is free from side effects. In other words, a data-flow operation is purely functional and produces no side effects such as the changes of memory words.

iii)No address required

Operands are directly passed as data-tokens representing values rather then addresses in the case of shared memory systems.

iv)Natural parallel control flow

The flow of control is tied to the flow of data and therefore the parallelism exhibited in the computation is represented in the control execution by data-flow [15].

1.5 DATA-FLOW AND COMPUTATION INTENSIVE ALGORITHM

Computation intensive algorithms or tasks require repeated arithmetic operations like add, subtract, divide and multiply, but the order of execution of these operations is not fixed by the task.

When the task or algorithm is of general purpose nature (e.g., sorting, searching and general information processing) then in order to utilize the concurrency in data-flow environment, the aim is to describe the computations to be performed without fixing there order of execution more than necessarily needed by the algorithm [6].

However, when the task or algorithm is computation intensive, the order of execution of instructions is not fixed by the algorithm. It is this attribute which makes computation intensive algorithms particularly suitable for data-flow environment [6].

Examples of computation intensive algorithms are digital signal processing algorithms. One such application of data-flow concepts in computation intensive algorithms is the evaluation of complex, long and tedious expressions.

1.6 OBJECTIVE OF THE THESIS

The objective of this thesis work is to evaluate computation intensive algorithms represented as computation expressions, by means of a computing machine. The computing machine would perform the computation in the data-flow environment. The data-flow expression evaluator would be able to evaluate two computation

schemas :

TYPE A)

STRAIGHT EXPRESSION: requiring single output with multiple inputs. An example of this form is

$$A^{1/2}x(BxC)+(Ax(B+C)^{2})-(B^{1/2}-A)$$

TYPE B)

ITERATIVE LOOP SCHEMA: multiple input in the form of operands and single output expression with iteration as in the following example

$$Y(I+1):=\{\{(B-Y(I))-(Y(i)^{1/2}xC)\}+(Y(i)xA)\}$$

for i:=initial value to final value

The corresponding data-flow graph representation of a straight expression is similar to Figure 1.1. The corresponding data-flow graph representation for the loop schema is shown in Figure 1.2. All operands, computations and outputs are in fixed-point , signed-magnitude integer arithmetic representation.

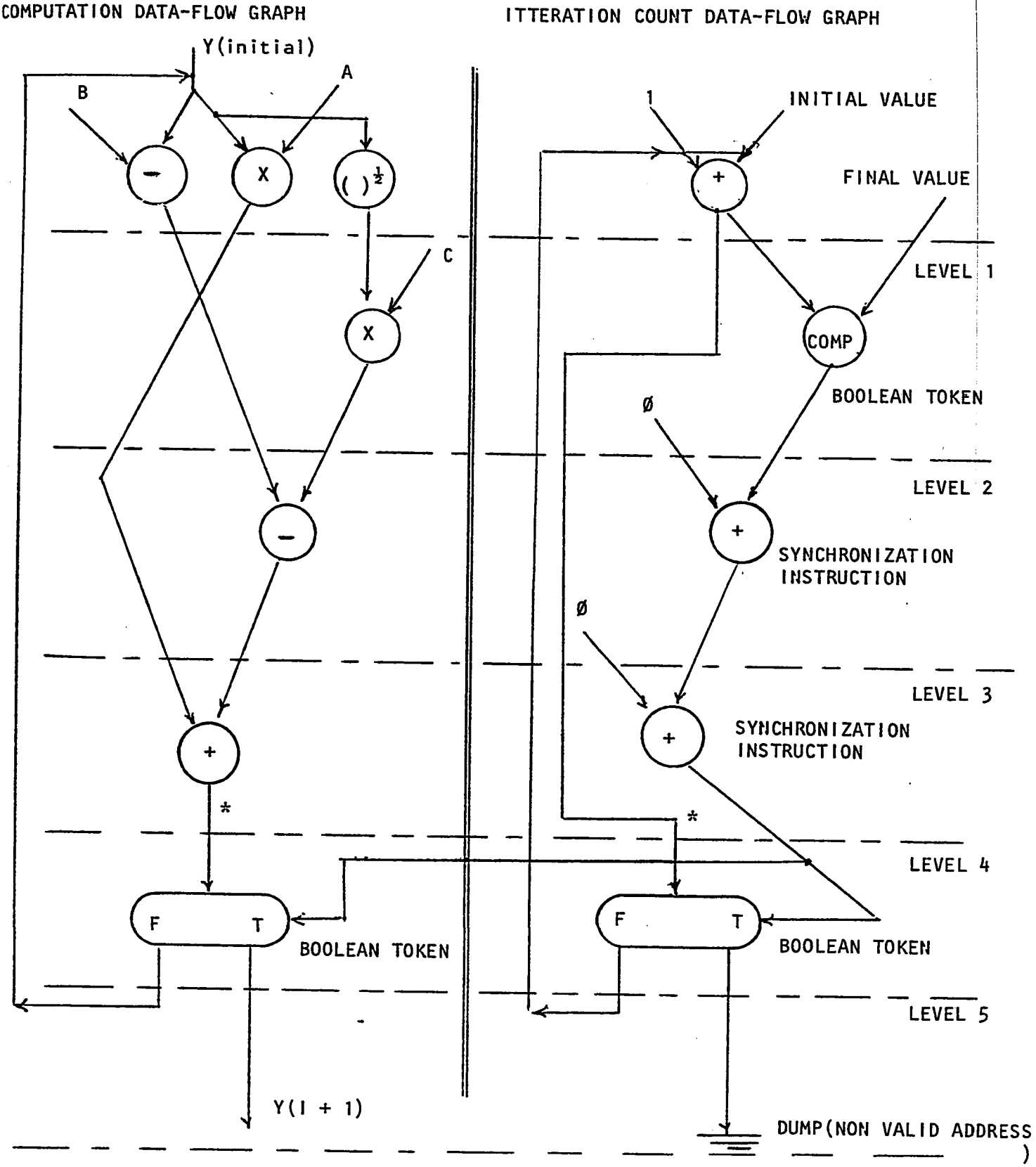


FIGURE 1.2: LOOP SCHEMA DATA-FLOW GRAPH

Literature review revealed that nine projects concerning the data-flow systems are currently underway in the globe. In these projects, a general-purpose computing system together with machine organization, implementation and functional programming language design (VAL, LUCID [15]) is envisioned. However, to exploit the data-flow concepts in a much smaller scale, in order to enhance the capabilities of the existing computers, had not been seen.

It is the purpose of this work to design an expression evaluator system within the framework of data-flow environment. Further, this work will develop a data-flow architecture which would perform the two types of computations indicated above. This architecture will then be mapped into a Very Large Scale chip design. Thus, this work would prove that the data-flow architecture is suitable to be mapped into VLSI circuits satisfying the methodology of VLSI design [9,10].

From this work, it would be seen that the data-flow computing architecture supporting the above mentioned computation intensive algorithms is completely recursive and modular. This architecture satisfies the criterion for good architecture for VLSI implementation [9,10].

1.7 OVERVIEW

The Data-Flow Expression Evaluator (DFEE) system would work as a special-purpose slave co-processor to the host computer. The user will input the expression from a program. This program will reside

on the host computer. The host computer will invoke the software resident on it to transform the computation expression into the required instruction format as required by the DFEE system. This software, known as the Optimization Compiler would do two functions. Firstly, it will take as an input the expression to be evaluated and map it into a corresponding set of instructions. Each instruction would be linked to the other instruction by the data-flow graph of the respective expression/computation. Secondly, the optimization compiler will partition these instructions into a sequence of instructions in such a way that maximium parallelism is gained from the DFEE system. This partitioning will be exclusively DFEE architecture-dependent. Thus, the optimization compiler would act as an interface between the programmer and the expression evaluator system. As a result, the programmer's task would reduce to simply inputting the expression.

Figure 1.3 represents the interaction of the optimization complier, the host computer and the data-flow expression evaluator system.

Chapter 2 describes the system architecture corresponding to a machine architecture which can execute data-flow graph represented computations. In order to verify the architrecture at a functional level, a detailed functional simulation is performed and the results are shown to be in agreement with the defined architectural functionality. In addition, it is shown in this chapter that the architecture is a suitable candidate for VLSI implementation.

Chapter 3 details the design of the sub-units used in the DFEE chip. It is seen from this chapter that all the circuits designed

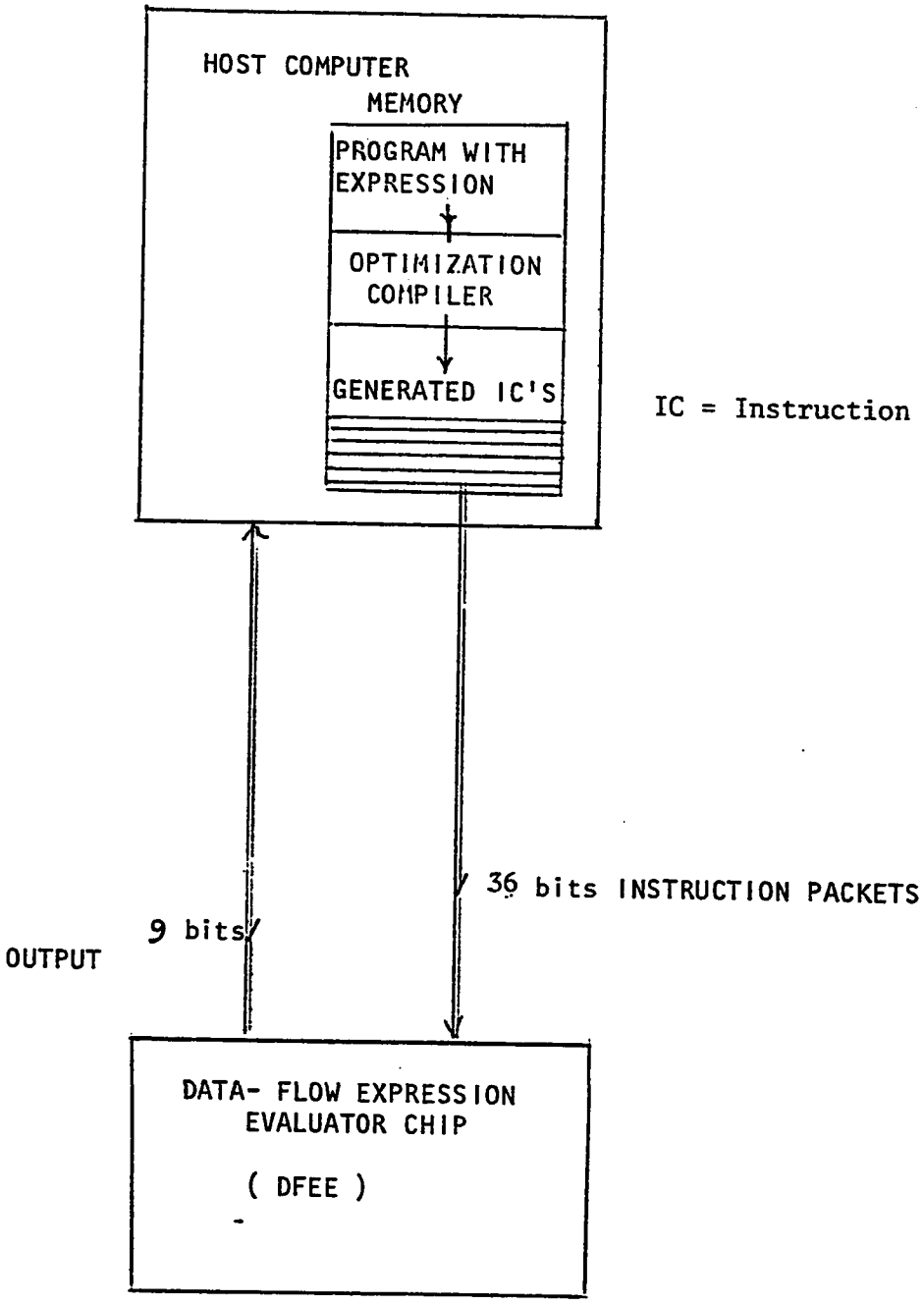


FIGURE 1.3: HOST COMPUTER AND DATA-FLOW EXPRESSION EVALUATOR CHIP INTERACTION

are suitable for VLSI chip implementation.

Further, in order to describe the designs, a language resembling PASCAL is used. This is done in order to make the description more structured and formal. In addition , this description can be used as the basis for programming in BELLE, the layout language which is embedded in PASCAL. Thus the DFEE chip design can be a suitable candidate for the application of BELLE Computer Aided Design (CAD) tool.

Chapter 4 details the processing element and the master controller design used in the DFEE chip. The processing element is the arithmetic logic unit of the chip architecture, while the master controller is the control unit of the DFEE chip. The designs are described in chapter 3.

Chapter 5 describes the definitaion and the algorithms for the Optimization Compiler. The optimization compiler is presented in the form of algorithms in order to make the implementation as much machine-independent as possible. Two of the most general algorithms are implemented in PASCAL as a guide for the user to implement the optimization compiler.

Lastly, some applications of the DFEE system are given in chapter 6. Further, some important conclusions for further research are also presented in this chapter.

Concluding, the end-product of this thesis would be a VLSI chip design which would serve as a slave co-processor to a host computer for computation speed-up and efficiency gains.

CHAPTER 2

DATA-FLOW EXPRESSION EVALUATOR (DFEE) ARCHITECTURE

In this chapter, interactions between computer architecture and VERY LARGE SCALE INTEGRATION (VLSI) will be given. In addition, this chapter will present the definition of the DFEE architecture. The functionality of this architecture will be described in terms of its sub-units.

The data-flow model represents a highly parallel computing system. It will be seen that the architecture defined by such a system is a suitable candidate for VLSI. This chapter will therefore conclude that the data-flow expression evaluator architecture can be implemented as a special purpose single chip in VLSI. This chip would be used as a co-processor [2] unit to enhance the performance of the general-purpose host computing system to which the DFEE is attached.

2.1 A SUITABLE COMPUTER ARCHITECTURE FOR VLSI [9,10]

A computer architecture which can be implemented into a single VLSI chip should have the following characteristics:

1) The system design should be made using systems partitioning such that modular components like registers, counters, decoders, demultiplexers, multiplexers are

employed extensively instead of the conventional AND, OR and NOT gate implementation. This approach would make the system architecture more modular and recursive.

2) If random logic has to be used using the conventional gates (NAND, NOR and NOT) then the logic should consist of only a few different types of cells, each cell being made up of the gates.

3) By building large and power consuming driver circuits onto the chip, it would be possible to drive the signal going through the various data paths as fast as minimum-size gates can drive them. In terms of design, this would mean that before inputting a signal through a pass transistor logic, a static inverter should be used and after getting out the signal from the pass-transistor logic, another such inverter should be used. In other words, it pays in terms of time to put inverter stages between logic modules [20].

4) For fast access times, the memory should be made static, unless stable operation is guaranteed to be reliable. Then in such case, dynamic memory can be used.

5) The architecture should have simple and regular data and control paths so that the cells can be connected by a network of local and regular interconnections (long distance or irregular communication should be minimized)[16].

2.2 THE BASIC ARCHITECTURE FOR DATA-FLOW EXPRESSION EVALUATOR SYSTEM

The basic architecture of the data-flow expression evaluator system is shown in Figure 2.1. Figure 2.2 shows its partitioning into sub-units. The phases shown are the non-overlapping phases transmitted throughout the DFEE chip by a global clock generator, represented by phi 1 and phi 2 (see Figure 2.3). The clock generator and the two-phase non-overlapping clock facility has been assumed a priori in this architecture and design.

Figure 2.2 shows the circular pipeline architecture of the DFEE system. Each sub-unit of the DFEE system receives the input at one phase and transmits the previous output at the complementary phase.

Figure 2.1 shows the interface between the DFEE system and the host computer. The DFEE system is a slave processor to this host computer. The communication between the host computer and the DFEE system is done through the master controller of the DFEE chip.

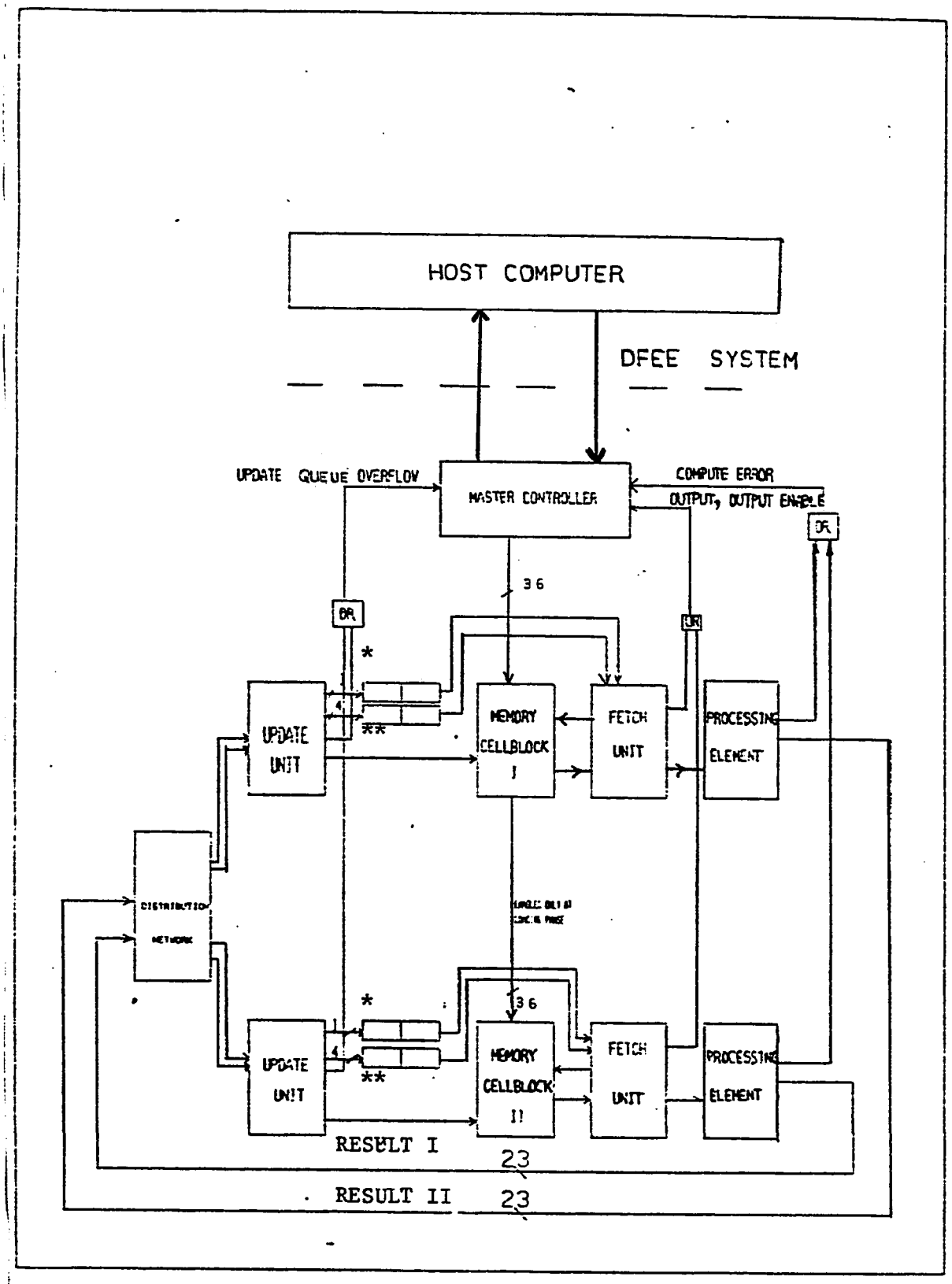
The master controller resides on the DFEE chip as an autonomous controller monitoring the operation and execution of the DFEE system.

2.2.1 LOADING PHASE

There are two phases of operation for the DFEE system :

a)Loading phase.

b)Execution phase.



* VALID ADDRESS QUEUE

**INSTRUCTION CELL (IC) ADDRESS QUEUE

FIGURE 2.1:DFEE BLOCK DIAGRAM (data-paths shown)

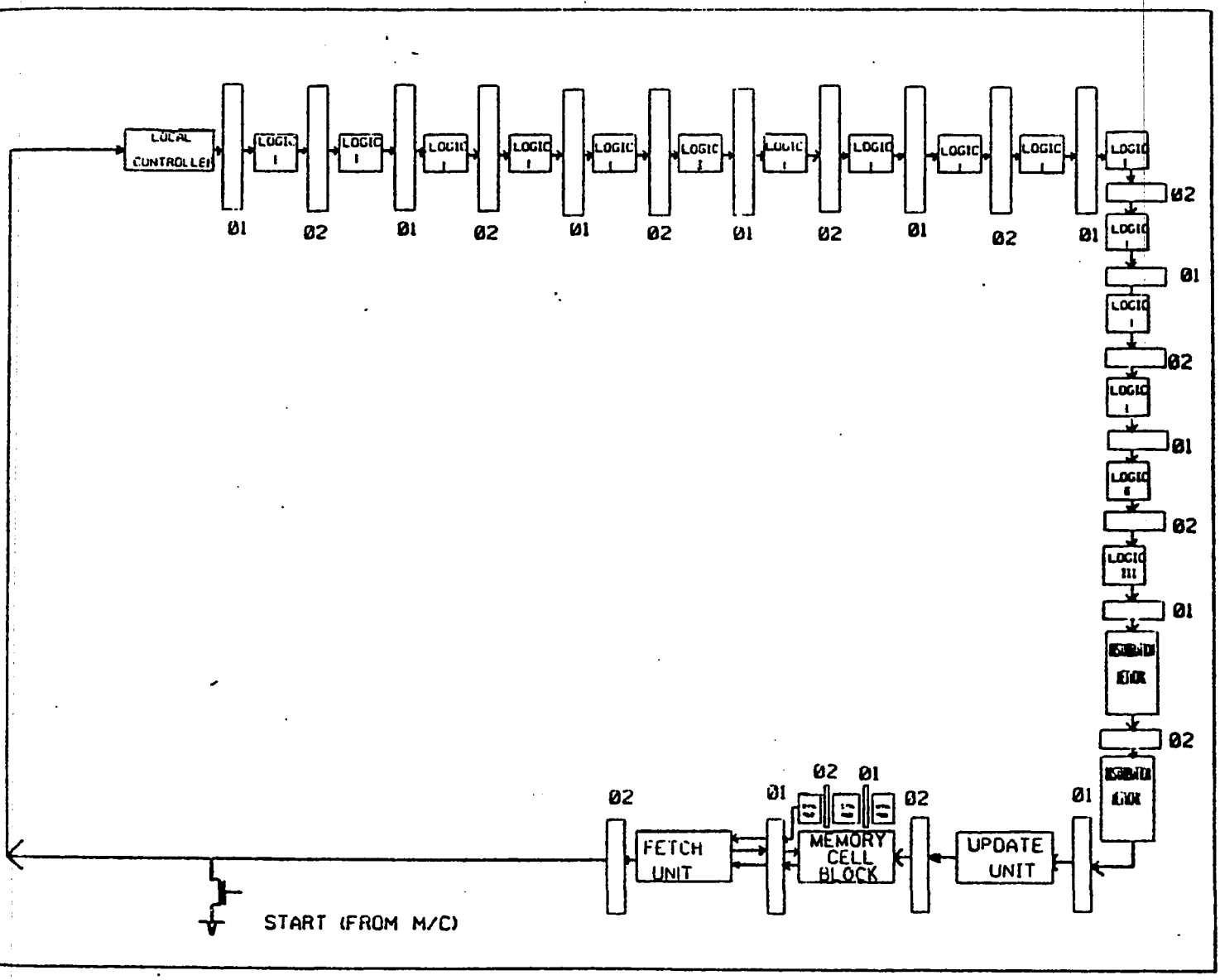


FIGURE 2.2:CIRCULAR PIPELINE ARCHITECTURE (only one path shown)

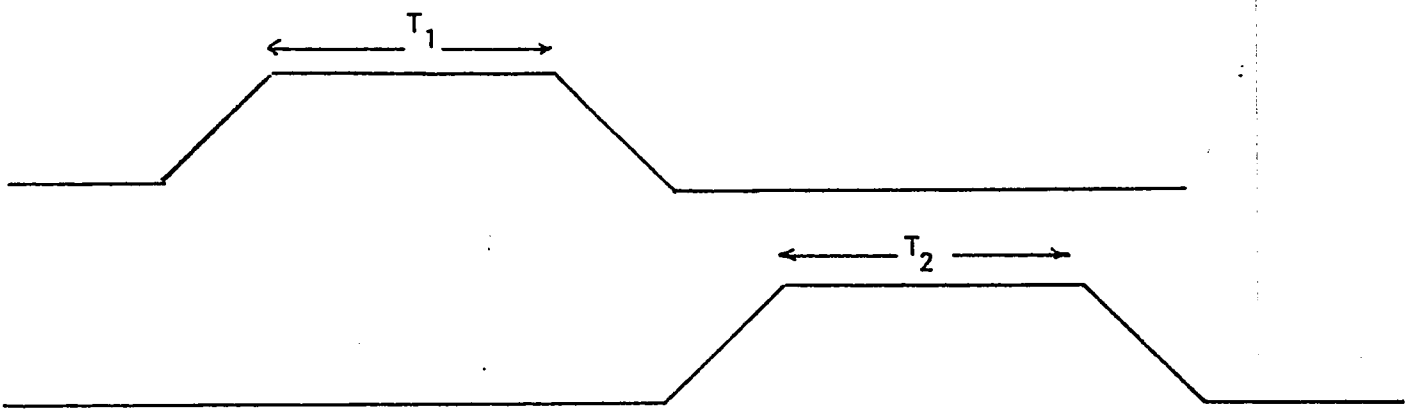


FIGURE 2.3: TWO-PHASE CLOCKING SCHEME

During the loading phase, the instructions from the host computer representing the computation are loaded into the DFEE system. The loading phase only involves the master controller and the memory unit while the rest of the sub-units are idle.

Each instruction consisting of 36 bits is shifted down into the DFEE system under the simple protocol between the master controller and the host computer. This protocol consists of examination of a single flag sent on a single bit line from the host computer. The flag (DATAIN) is examined by the master controller. If it is high then the loading phase is started. A low signal on this flag tells the master controller that all the data has been input into the DFEE chip from the host computer and the master controller can go ahead with the start of execution phase. The instructions are loaded into the memory unit of the DFEE system in a parallel-shift-in manner. The memory unit acts as a one-direction stack during the loading phase. A snap shot of the loading phase is shown in Figure 2.4.

Once all the instructions from the host computer are loaded into the DFEE system, then the host computer indicates this fact to the master controller. The master controller, then gives the signal to put the DFEE system into the execution phase.

2.2.2 EXECUTION PHASE

In the execution phase, all the sub-units become active and complete the circular pipeline shown in Figure 2.2. The master controller does not take part in the execution phase.

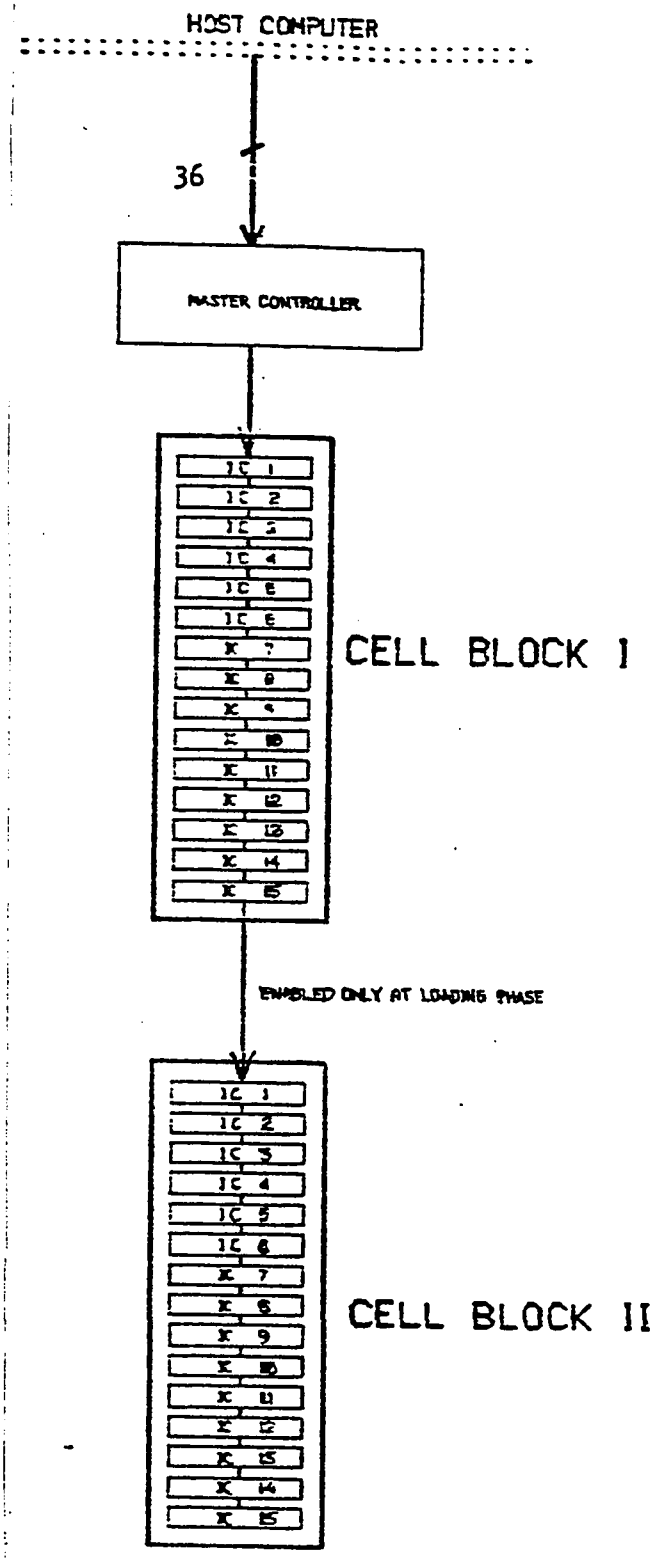


FIGURE 2.4:SNAP SHOT OF THE DFEE LOADING PHASE

The master controller monitors the DFEE system for possible error condition (e.g divide-by-zero). Once the computation is successfully completed, the master controller enables the output to the host computer.

2.2.3 INSTRUCTION AND RESULT FORMATS

There are two forms of data fields in the DFEE system :

i) INSTRUCTION FORMAT

The instruction is loaded from the host computer into the DFEE memory unit. The instruction field consists of 36 bits. The format of the instruction is shown in Figure 2.5. The description of the instruction format follows.

Field 1: Opcode

This field is 4-bit long and contains the opcode. The opcodes used by the DFEE system are as follows :

OPCODE	HEX CODE
NOP	00H
ADD	01H
SUBTRACT	02H
MULTIPLY	03H
DIVIDE	04H
SQRT	05H
SQR	06H

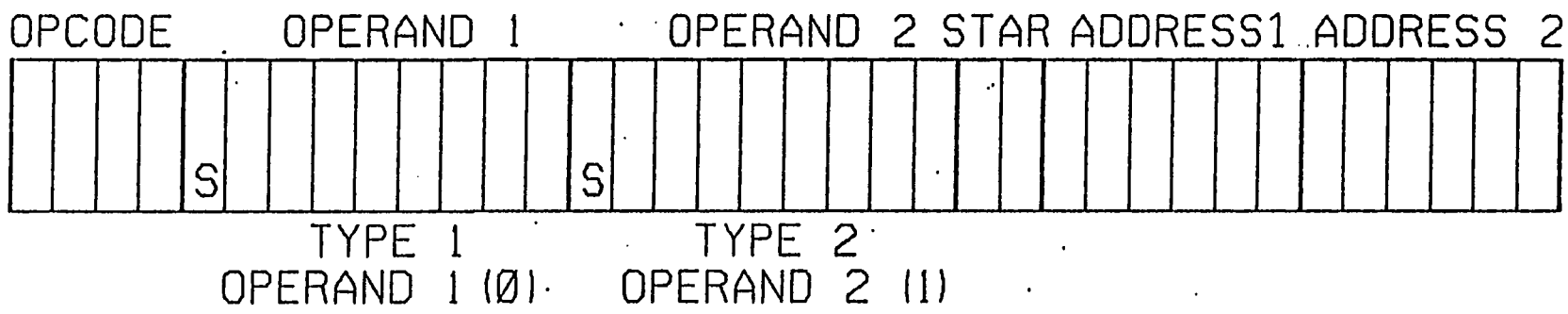


FIGURE 2.5:FORMAT OF INSTRUCTION

COMP	07H
LOOP	08H
FIN(ish)	09H

Field 2: Operand 1

This field contains operand 1. Operand 1 is a 9-bit signed-magnitude integer number. Bits 0...7 contain the magnitude part of the operand 1 while bit 9 stands for the sign bit (1 represents a negative number, 0 represents a positive number).

Field 3: Operand 2

This field contains operand 2 which is also a signed-magnitude integer number.

In order to represent a blank operand 1, operand 2, a value of 100000000 (100 Hex) is used. This value is chosen because it is not used in the number system to represent any integer (integer '0' is represented as 000000000). Thus the instructions formed by the host computer will take this into account and put a 100 Hex number instead of operand 1 and/or operand 2 to represent a blank operand.

Field 4: Star

This field contains the star field. It is a 2-bit field. The purpose of the star field is to distinguish the four possibilities

associated with field 6 and 7 (i.e Address 1 and Address 2 fields). The description of the four possibilities follows :

00:(A)

Take the address 1 field as the valid destination address of the result.

01:(B)

Take the address 1 and address 2 field as the valid destination address of the result. In this case, the result will be sent to two addresses in the memory.

10:(C)

Take the address 1 field as the valid destination address of the result (same as A but used only with the LOOP schema).

11:(D)

Take the address 2 field as the valid destination address of the result.

Field 5: Address 1

This field contains address 1. This address is the physical

address through which each field in the instruction cell is accessed. Instruction cell is the physical storage in the memory unit where the instruction is stored. One instruction is stored in one instruction cell. Address 1 consists of the following sub-fields:

CB: This is a single bit. It distinguishes the two memory cell blocks shown in Figure 2.1

IC: This field is 4-bit long. It contains one of the 15 addresses of the cell block instruction cells,.i.e., each cell block contains 15 instruction cells. The binary vector 0000 (b) is not used as an address therefore the code is reduced to 15 possible vector values,.i.e., 1,2,3...15 (decimal).

OPR: In each of the instruction cells, either operand 1 or operand 2 can be addressed. In order to distinguish between these operands, a single bit is used.

To represent one of these fields, the following notation will be used :

Operand field of address 1. This will be described as:

ADDRESS 1.OPR

The complete address 1 will be represented by :

ADDRESS 1(.CB.IC.OPR)

Field 6: Address 2

This field contains address 2. The description of the sub-fields is the same as Address 1.

ii) RESULT FORMAT

The result field contains the 9-bit output result from the processing element computation. This result field contains 23-bits of data (Figure 2.6). The 9-bit result is destined to operand 1/ operand 2 field in one of the instruction cells inside one of the memory cell blocks. The Star , Address 1 and Address 2 fields specify the destination of the 9-bit result .

This result goes from the processing element into the distribution network, update unit and finally into the memory cell block instruction cell. The description of the result format follows:

Field 1: Result

The result represents the 9-bit result value which is to be transfered to the corresponding memory cell block instruction cell.

Until all the instructions are executed , the result contains the partial result flowing in the DFEE system. Once, all the instructions inside the memory cell blocks are executed , this result contains the final result of the computation.

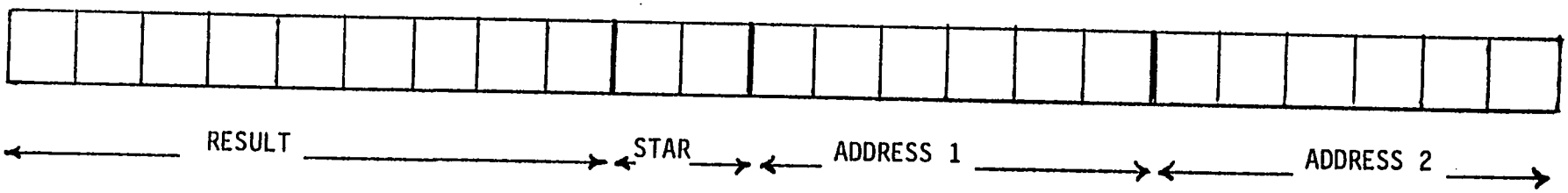


FIGURE 2.6:RESULT FIELD FORMAT

Field 2: Star

This field represents the Star field as described in the instruction format defination.

Field 3: Address 1

This is the Address 1 field as described in the instruction format defination.

Field 4: Address 2

This is the Address 2 field as described in the instruction format defination.

2.3 FUNCTIONAL DESCRIPTION OF THE EXECUTION UNIT

The DFEE system can be divided into two parts (Figure 2.7):

i) THE CONTROL UNIT: The control unit of the DFEE is the master controller. It only takes part in the loading phase and monitors the execution phase .

ii) THE EXECUTION UNIT: The execution unit consists of the sub-units as shown in Figure 2.1. These sub-units become active in the execution phase. Therefore all these sub-units form an autonomous system until either the computation is

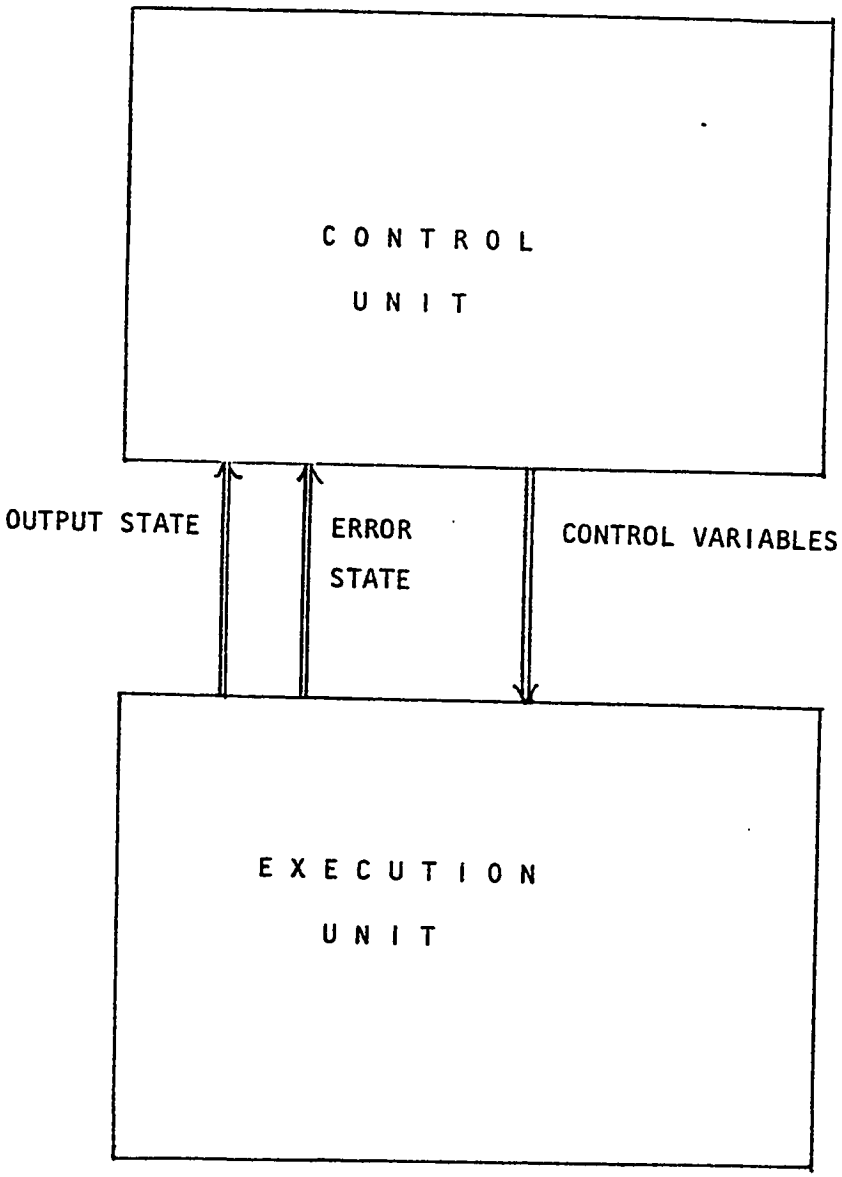


FIGURE 2.7:DFEE BLOCK DIAGRAM

mapped into an error state or a correct output state . Once, when one of these states is reached by the execution unit, then the control is transfered to the master controller. This architecture is called Circular Pipelined Architecture [5].

The execution unit consists of the following sub-units :

i) Distribution Network.

ii) Update Unit.

iii) Memory Unit.

iv) Fetch Unit.

v) Processing Element.

The functional description of the sub-units follows;

2.3.1 DISTRIBUTION NETWORK

Figure 2.1 shows the distribution network. The distribution network receives two results (RESULT I and RESULT II) from the two processing elements. The function of the distribution network is to examine these result field addresses (the cell block CB field) and accordingly route the 9-bit output results to the two cell blocks. Therefore the function of the distribution network is that of a switching network.

Depending on the addresses of the two results received by the distribution network, there are 16 routing possibilities (2 bits from Star I and 2 bits from Star II fields). All of these routing

possibilities are used in the architecture. However, these 16 routing possibilities can be grouped into the following routing strategies:

* Result I is destined for cell block I and result II is destined for cell block II.
* Result I is destined for cell block I as well as cell block II. Result II is destined for cell block II.
* Result I is destined for cell block I. Result II is destined for cell block II as well as cell block I.
* Result I is destined for cell block II only. Result II is destined for cell block II. There is no result destined for cell block I.
* Result II is destined for cell block I only. Result I is destined for cell block I. There is no result destined for cell block II.
* Result I is destined for cell block I as well as cell block II. Result II is destined for cell block I as well as cell block II.
* There is no result destined for either cell block I or II. The distribution network receives a cell block address of 0. It sends the 9-bit result to cell block I. However, there is no IC (Instruction cell) address of 0000B and in this way no data is written into either cell block I or cell block II instruction cells. For this case, the address 1 and address 2 fields should be an all zero vector of length 6-bits.

These routing strategies in the distribution network are handled by the inspection of the star bits of the respective result I and result II fields.

There can be two results destined for the same cell block. Therefore, two results going into the distribution network can produce four results - two results going to cell block I and two results going to cell block II. In the case, that the result is not destined to any cell block, the result is grounded (zeroed) by the distribution network.

The cell block address field (CB-1 bit) is discarded by the distribution network for both address 1 and address 2. This is because once the destination cell block is found by the distribution network, there is no need to carry its 1-bit address along with the IC and OPR addresses.

2.3.2 UPDATE UNIT

The update unit writes the result computed from the processing element into the memory cell blocks. The process of writing in the instruction cell of the memory cell block updates the instruction cell.

The update unit has the following function:

a) The update unit receives the two results from the distribution network. Depending on the addresses of the result fields, it may need to update only one result into the cell block which is associated with it (Figure 2.1), copy the same result into two different instruction cell locations in the same cell block or update two different results in two different instruction cell locations. This function is performed by the update unit by inspection of the Star ,

Address 1 and Address 2 bits of the respective Result field I and II. The updating is performed by the update unit by giving appropriate write addresses to the memory cell block for the 9-bit result.

b) Whenever there is an update operation performed by the update unit, this has to be communicated to the fetch unit so that it can pick the latest updated instruction cell from the cell block. Thus, as soon as the update unit writes the result/results into the cell block memory , it pushes the address of the instruction cell just updated into the Instruction Cell (IC) Address Queue as shown in Figure 2.1. Further, in order to communicate to the fetch unit that a valid address resides on the instruction cell address queue, it puts logic-1 into Valid Address Queue.

These two queues , IC Address Queue and Valid Address Queue are shown in Figure 2.1. The Valid Address Queue is a single bit queue while the IC Address Queue is a 4-bit queue. The length of both the queues are 2, consisting of two shift-registers.

The IC Address Queue and the Valid Address Queue are in synchronization; that is, if the output of the Valid Address Queue is logic-1 then the same output of the 4-bit IC Address Queue is a valid instruction cell address. The need for Valid Address Queue and the IC Address Queue was found from the simulation results (Section 2.5.2). Three cases of IC address queue loading are in order:

* In the case that only one result is updated, the IC address is put at the bottom of the queue. The fetch unit will fetch this address not during this phase but with a delay of one phase due to the queue.

* In the case that two results are written into the memory cell block, both the IC addressess are loaded into the IC Address Queue. Correspondingly, the Valid Address Queue is loaded with the true values. This way, the IC Address Queue and the Valid Address Queue is full. With this state, the fetch unit picks out the topmost or the right-most address value. However, if the next set of results again update the cell block in two IC addresses, then the previous IC address will be lost (See Figure 2.8). In the case of loosing the address, the update unit does not give any warning. This can be avoided by extending the IC Address Queue and the Valid Address Queue. However, this leads to degregation of time delay because more clock pulses are required. Thus a trade-off is chosen where the instruction cells are placed in such a manner that only two instructions are updated in the same cell block when the IC Address Queue is empty. This requirement is put onto the generation of the instructions by the host computer (see section 2.5.3).

* If three or more instruction cells are written in the same cell block, the update unit gives out a warning to the master controller that the IC Address Queue and the Valid Address Queue have overflown. The master controller then

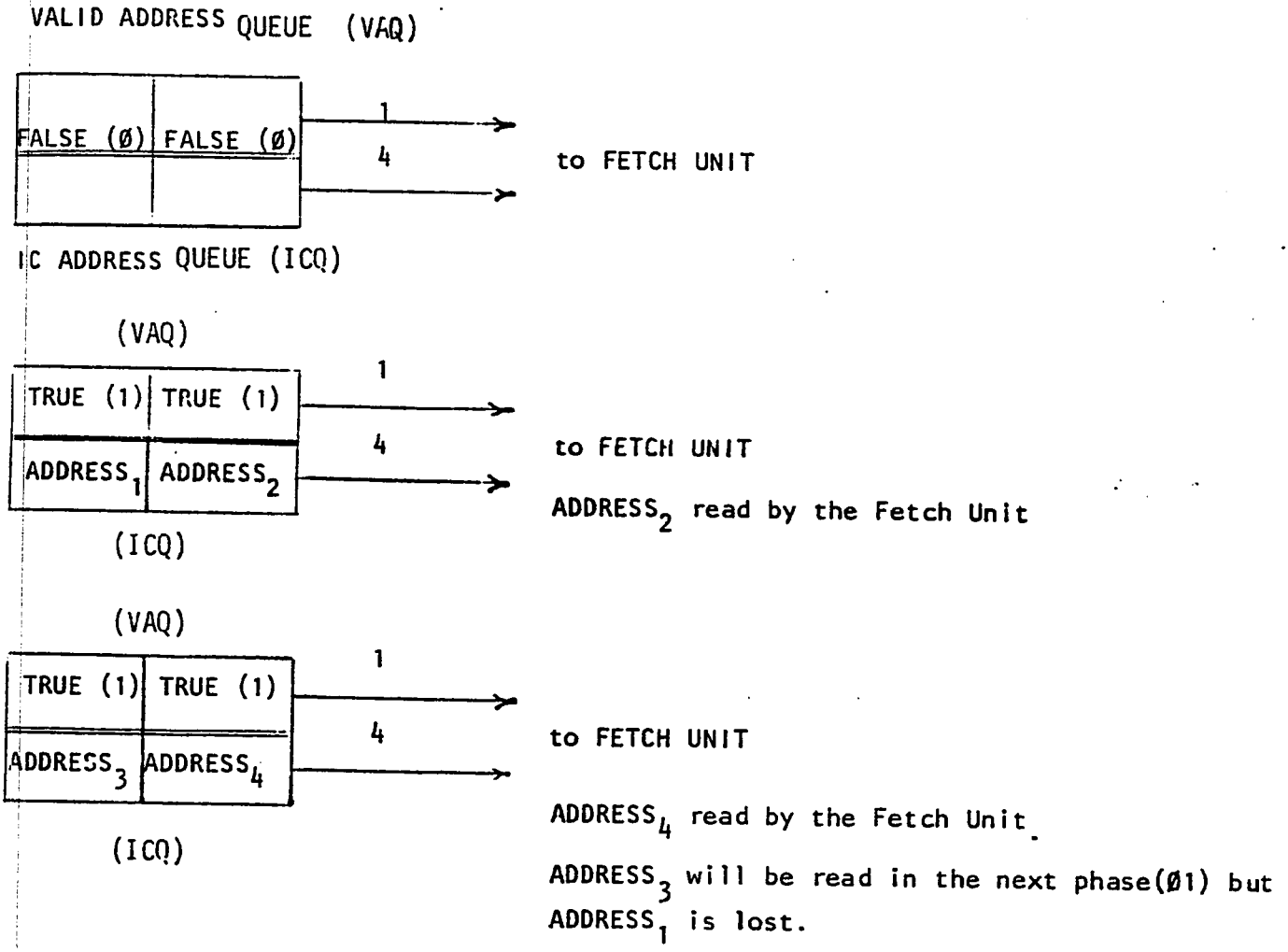

VALID ADDRESS QUEUE (VAQ)
FALSE (Ø)
FALSE (Ø)
1
4
to FETCH UNIT
IC ADDRESS QUEUE (ICQ)
(VAQ)
TRUE (1)
TRUE (1)
ADDRESS1
ADDRESS2
1
4
to FETCH UNIT
ADDRESS2 read by the Fetch Unit
(ICQ)
(VAQ)
TRUE (1)
TRUE (1)
ADDRESS3
ADDRESS4
1
4
to FETCH UNIT
ADDRESS4 read by the Fetch Unit.
ADDRESS3 will be read in the next phase(Ø1) but
ADDRESS1 is lost.
(ICQ)

FIGURE 2.8

aborts the whole operation and warns the host computer by disabling the clock (phi 1,phi 2). The master controller does this by grounding all the data-paths in the circular pipeline and putting the error flag high for the host computer to detect.

The minimum queue length for Valid Address Queue and the IC Address Queue should be equal to the number of cell blocks. For the case shown in Figure 2.1, the minimum length is 2.

2.3.3 MEMORY UNIT

The memory unit of the DFEE system consist of two cell blocks. Each cell block is linked with the corresponding update and fetch unit (Figure 2.1). Cell block I is connected with cell block II in order to load the instructions from the host computer at the loading phase (Figure 2.4). Therefore, the memory unit can be seen as a one-directional stack of depth 30 in the loading phase.

Once in the execution phase, the cell block is written into by the update unit at phi 2 and read from by the fetch unit at phi 1. Each cell block is divided into 15 instruction cells. Therefore a maximum of 30 instructions can reside into the DFEE system. If more instruction cells are required, the address lines of the cell blocks can be extended without incurring any increase in the complexity of the circuitry. In the case that there is a read or write address of 0000B, the memory cell block is left as it is because the IC 0

address does not exist.

2.3.4 FETCH UNIT

The function of the fetch unit is to fetch the most recently updated instruction cell from the cell block memory. This information is obtained from the IC address queue and the valid address queue.

The fetch unit will take the valid address queue signal with priority. This priority is needed because in some instances, there can be two "full" instructions ready for fetching. The "full" implies that operands are available for computation.

One case is that the full instruction can already exist in the cell block memory. The other case is that the instruction would be updated by the update unit and whose address has been conveyed to the fetch unit by the IC address queue.

If the valid address signal is not high (logic-0), then the fetch unit starts fetching the full instruction cells in the order of priority IC1, IC2, IC3,...,IC15. In this case, once the IC is fetched, it will not be fetched again unless the update unit gives the signal through the two queues.

In the begining, there will not be any update due to the pipeline filling of the processing element (section 2.3.5). Therefore, this mechanism will enable the fetch unit to fetch the full instruction cells. These instruction cells would be the ones with constant literals.

The fetch unit is not permitted to fetch the same full instruction cell twice in a row. This is to prevent the case when two results are input into the same instruction cell. The instruction cell becomes full. However, the IC address queue contains the same address twice because the instruction cell is written twice with two results, one for Operand 1 and one for Operand 2. The fetch unit will fetch this instruction cell from the cell block indicated by the IC Address Queue. However, in the next cycle, the fetched instruction cell address will still reside on the IC address queue. Therefore, in order to avoid fetching the same instruction cell twice in a row, the fetch unit preserves the fetched instruction cell address in its local register. This observation was made from the simulation program (see section 2.5).

2.3.5 PROCESSING ELEMENT

The processing element is the arithmetic unit of the DFEE system. It receives the instruction consisting of the operand(s) and the operation from the fetch unit. It performs the operation of ADD, SUBTRACT, MULTIPLY, DIVIDE, SQUARE-ROOT and SQUARE on the operand(s) in fixed point sign magnitude integer arithmetic.

For the case of loop schema, it does special functions for the LOOP and COMP(are) opcode instructions. For the compare instruction, it puts a special result in Operand 1. The LOOP instruction uses the result of the COMP instruction.

For the LOOP instruction, Ooperand 2 has to be routed

conditionally to one of the addresses specified in the address fields. The processing element inspects Operand 1 and modifies the Star bits in the address part of the result field to do the above.

The loop and compare instructions are related as described above. However, the compare instruction will always supersede the loop instruction in execution.

The processing element is pipelined with 18 stages. The processing element consists of four sub-units: local controller, logic unit I, II and III. These sub-units are divided as stages of the pipeline as shown in Figure 2.2.

2.4 EXTENSION OF THE DATA-FLOW EXPRESSION EVALUATOR ARCHITECTURE

From Figure 2.1, it is seen that the architecture satisfies the VLSI CRITERIA mentioned in section 2.1. It can be seen that the DFEE architecture is modular and recursive. In order to extend the architecture in terms of instruction cells, more address lines to the memory cell blocks can be added without doing any other design modifications.

The DFEE architecture can be extended. For example, it can be extended to four cell blocks and correspondingly to four update units, four fetch units, four processing elements and one distribution network. The distribution network will always remain one in the extension.

In the case of four cell blocks, the update unit will need to update four results (as a worst case). Consequently, the IC address queue and the valid address queue lengths must be extended to four. In the original case (.i.e, two cell blocks), sixteen possibilities exist; 4 possibilities from each star bits value lead to (4 x 4) sixteen data-selector units.

In the extended case (.i.e, four cell blocks), since four results enter into the update unit, each having 2 bit star value, the result routing possibilities will increase to (4 x 4 x 4 x 4) two hundred fifty six (256) data-selectors.

Thus the complexity of the distribution network and the update unit will increase exponentially. For the fetch unit and the processing element, the complexity will remain the same, since these units deal with the same single instruction.

The attributes of the circular pipeline architecture make the DFEE architecture an ideal candidate for VLSI implementation. The efficiency of the circular pipeline architecture is increased most when the processing element is also pipelined [9,10]. This has been done in the design where the processing element has 18 pipelined stages.

2.5 FUNCTIONAL VERIFICATION AND SIMULATION OF THE DFEE ARCHITECTURE

The functional verification was done by simulating the DFEE architecture for both straight expressions and loop schema. The functional verification at this stage was necessary for two reasons.

Firstly, it would verify the architecture at the functional level. Secondly, it would help in the design phase of the DFEE chip.

2.5.1 SIMULATION ENTITIES AND RELATIONS

The architecture represented a non-von-Neumann computing system. It exhibited inherent parallelism and novel computing concepts. Therefore, conventional hardware design languages of simulating the architecture (at the register transfer level) like AHPL would likely make the functional representation of the DFEE architecture more difficult.

Thus a simulation program was written in a high level language-PASCAL. PASCAL was chosen for two reasons. Firstly, it is a very structured language and would represent the functionality of the architecture well. Secondly, since the simulation was done from scratch, a language was needed which was rich in data-structures; and PASCAL was a resonable choice.

Each cell block was represented as an array of record. The array element was the instruction cell. Different fields in the instruction cell were represented as record fields. The simulation program has provisions for extension of the architecture in terms of the cell blocks, instruction cells and the queue lengths.

Appendix A gives the simulation program and the listing of the output for the following type A straight expression :

output:=(a-b) x (a+b)

Appendix B shows the simulation program and the listing of the output for the following type B loop schema:

y(I+1):= Y(I) x 10 + 10

I:=1 to 3

Both the programs contain the instruction cells formed as they would appear inside the cell blocks (the fields have been coded for clarity in reading).

2.5.2 RESULTS OF THE SIMULATION

The following observations were drawn from the simulation program:

i) The distribution network can be represented as a switching network.

ii) The update unit can be represented as a functional unit and also as a register transfer unit.

iii) The fetch unit can be represented as an algorithmic state machine which executes register transfer operations and also as a functional unit which maps input into an output state.

iv) The fetch unit requires local working units for storage and counting, .i.e,a 4-bit register and counter.

v) The processing element is purely a functional unit which maps an input vector into an output vector.

vi) The IC Address Queue and the Valid Address Queue lengths should be greater than or equal to the number of the cell blocks.

It was found that the architecture defined for the DFEE was functionally correct and working for both the loop schema and the straight expression computations.

2.5.3 TRADE-OFFS AND CONSTRAINTS

The following trade-offs and constraints were observed from the simulation:

1) In the case that the update unit updates or writes two results in the same cell blocks, there is a possibility of overloading the IC Address Queue and the Valid Address Queue. This problem can be avoided if the IC Address Queue and the Valid Address Queue length is increased to twice the number of cell blocks,.i.e, 4. However, this severly degrades the time performance since more time delay is involved. Thus the following constraint is put on the instruction placing in the instruction cells.

   The instructions should be placed in the cell blocks such that the IC Address Queue is filled with two different addresses simultaneously (in the case of two updates) if and only if the IC Address Queue was empty previously.

2) For the case of LOOP SCHEMA, in addition to the constraint mentioned in (1), the following constraint should be satisfied:

   Two data-flow graphs are used to execute the loop schema- one data-flow graph for the computation itself, the other data-flow graph for counting the iteration. There has to

be synchronization between the boolean token passed between the two data-flow graph executions. This synchronization is established by inserting null synchronization instructions if needed (see Figure 2.9).

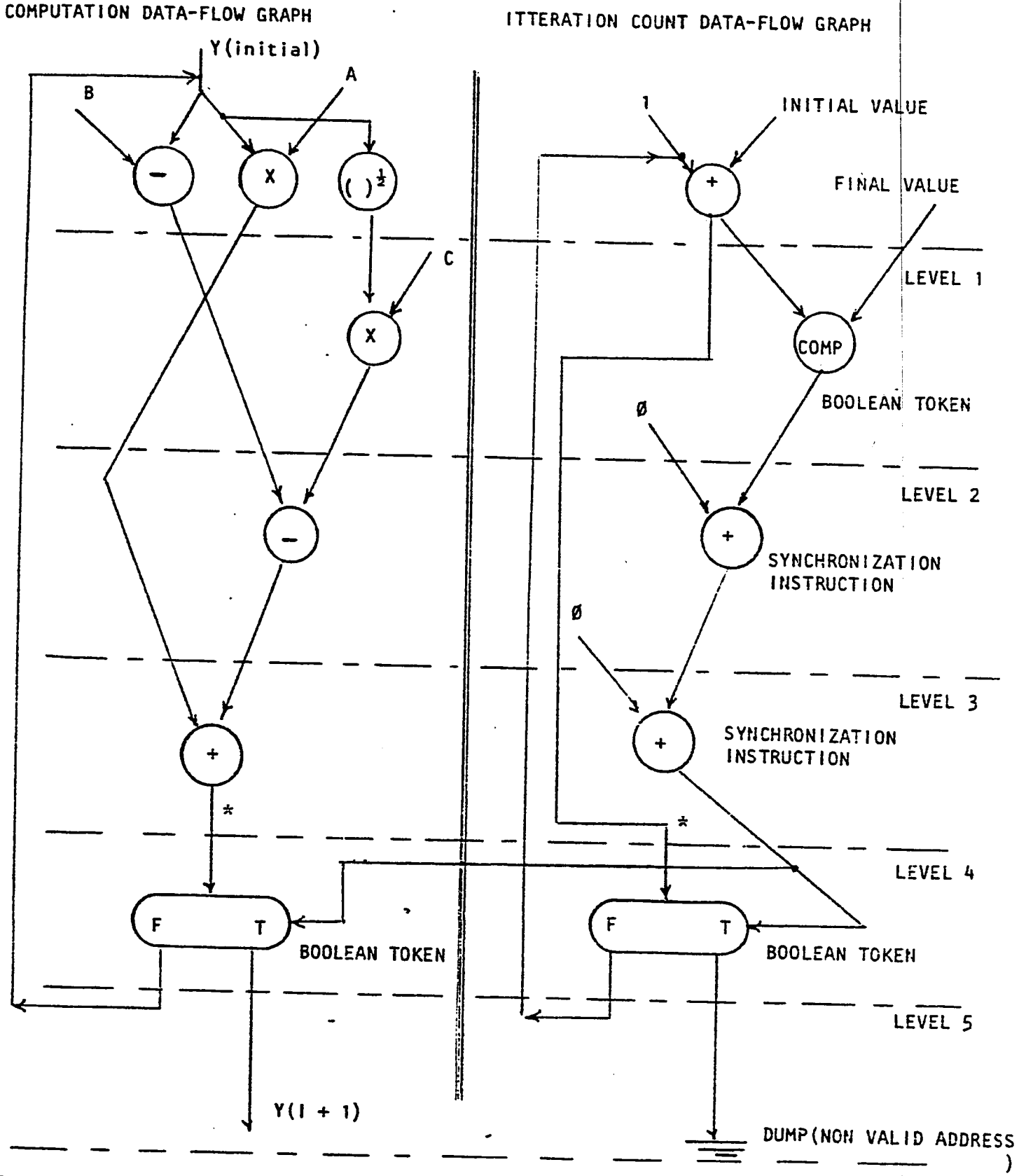


FIGURE 2.9:DATA-FLOW GRAPH FOR LOOP SCHEMA

CHAPTER 3

DESIGN AND IMPLEMENTATION

This chapter discusses the design and implementation of the distribution network, the update unit, the memory unit and the fetch unit. The design is based on the VLSI criterion described in section 2.1.

Each design is made regular. More regular structures like tree structured multiplexers and demultiplexers are used extensively in the design to select the data paths and to generate the enable signals. The basic structure for the demultiplexer is shown in Figure 3.1. Figure 3.2 shows the basic structure for the multiplexer.

3.1 OVERVIEW OF DFEE SYSTEM ARCHITECTURE

Figure 3.3 shows the system architecture of the DFEE system. The operation of the DFEE consists of the loading phase and the execution phase. In the loading phase, the instructions representing the computation are loaded into memory cell block I and II. A total of 30 instructions can be loaded into the memory cell block of the DFEE system.

All the 30 instruction cells have to be filled in by the host computer in the loading phase. All the 30 instruction cells in the DFEE

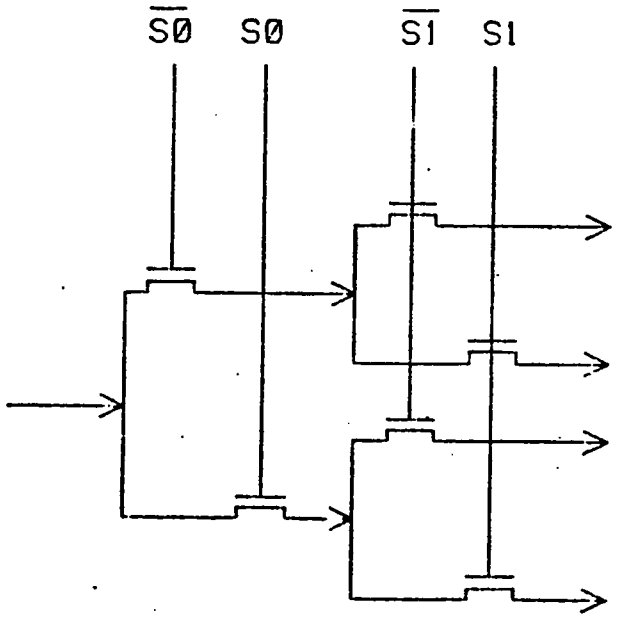


FIGURE 3.1: 1-BIT DEMULTIPLEXER WITH 2 CONTROL BITS

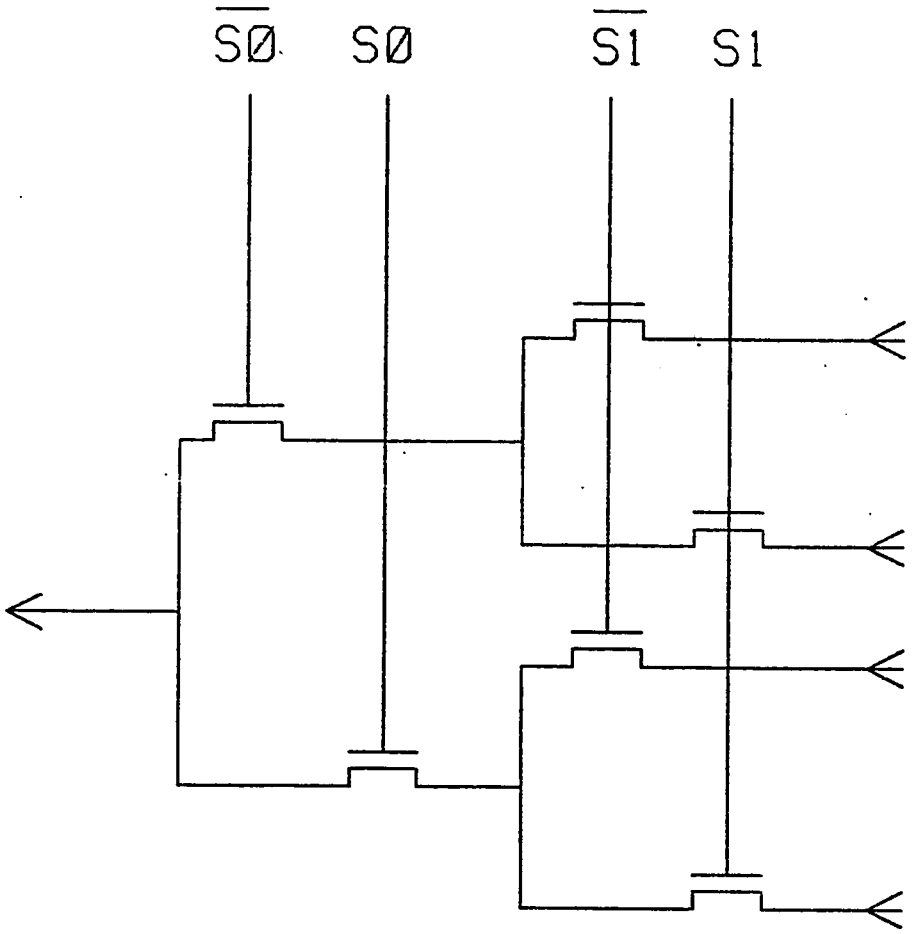
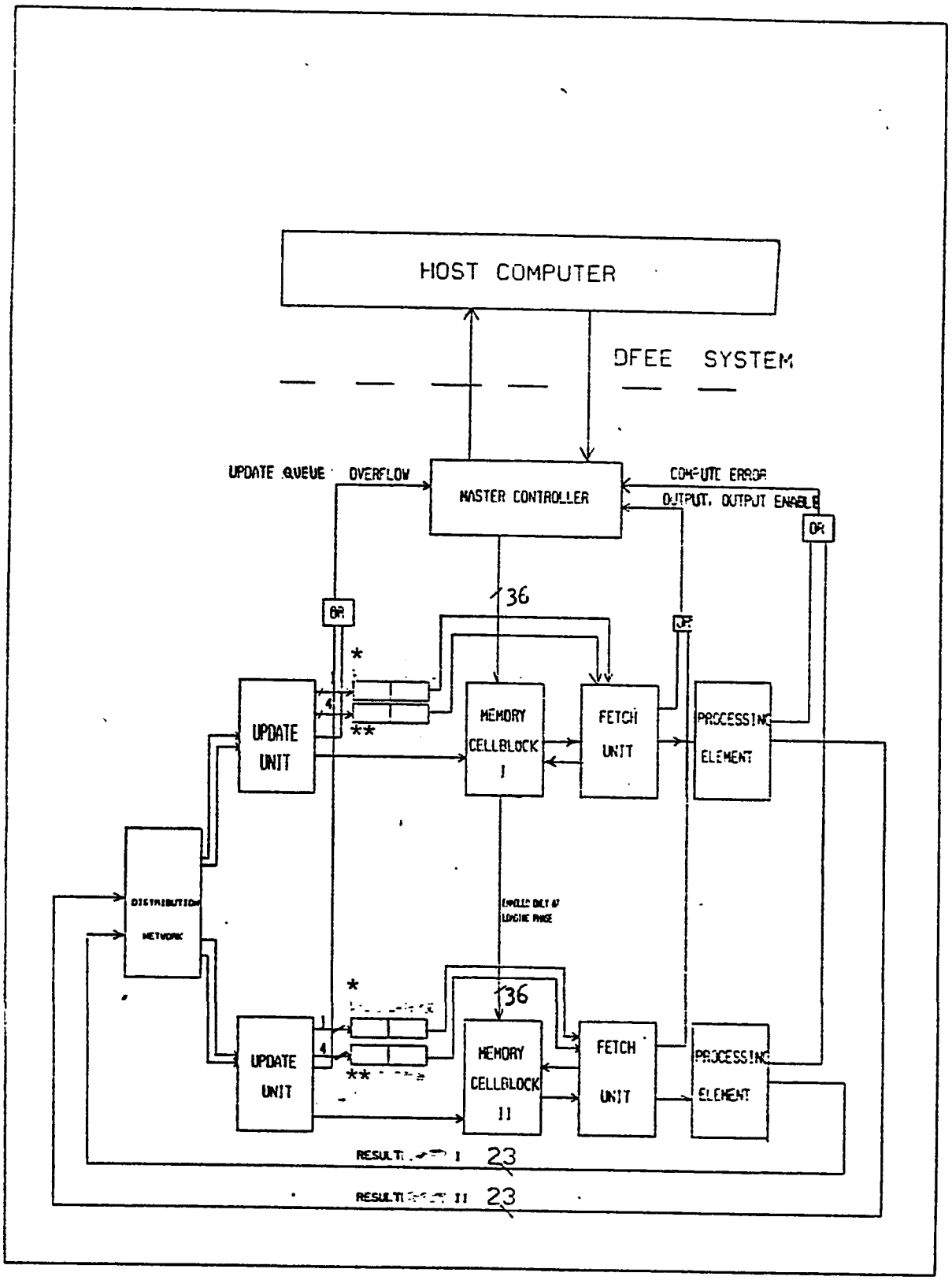


FIGURE 3.2:1-BIT MULTIPLEXER WITH 2-BIT CONTROL



* VALID ADDRESS QUEUE
**IC ADDRESS QUEUE

FIGURE 3.3:DFEE ARCHITECTURE-BLOCK DIAGRAM WITH DATA-PATHS

hardware have to be initialized. For the loading, no address is required for accessing the instruction cells in the memory cell blocks. The instructions are pushed down from top to bottom into the memory cell blocks I and II. Together, these cell blocks form a one directional stack of length 30 in the loading phase (Figure 2.4).

The loading phase is monitored by the master controller. The master controller receives a signal from the host computer to begin the loading phase. In the loading phase, all the sub-units of the DFEE system are idle,.i.e, no reading/writing operation is done on the memory cell blocks; the circular data-paths are grounded (Figure 3.4).

The end of the loading phase is indicated by the host computer to the master controller by a flag. The master controller then puts the DFEE system into execution phase by enabling the data-paths of the circular pipeline (Figure 3.4).

In the execution phase, all the sub-units become active. Each instruction goes through the fetch unit, the processing element, the distribution network, the update unit and into the memory cell block, either cell block I, II or both. The instruction contains opcode and operand/s. The operand/s and the opcode are used by the processing element to compute the result of the operation.

The result is distributed to one or both of the cell blocks by the distribution network. The update unit writes the result into the instruction cell. The fetch unit picks out the most recently updated instruction cell. This information is conveyed to the fetch unit by

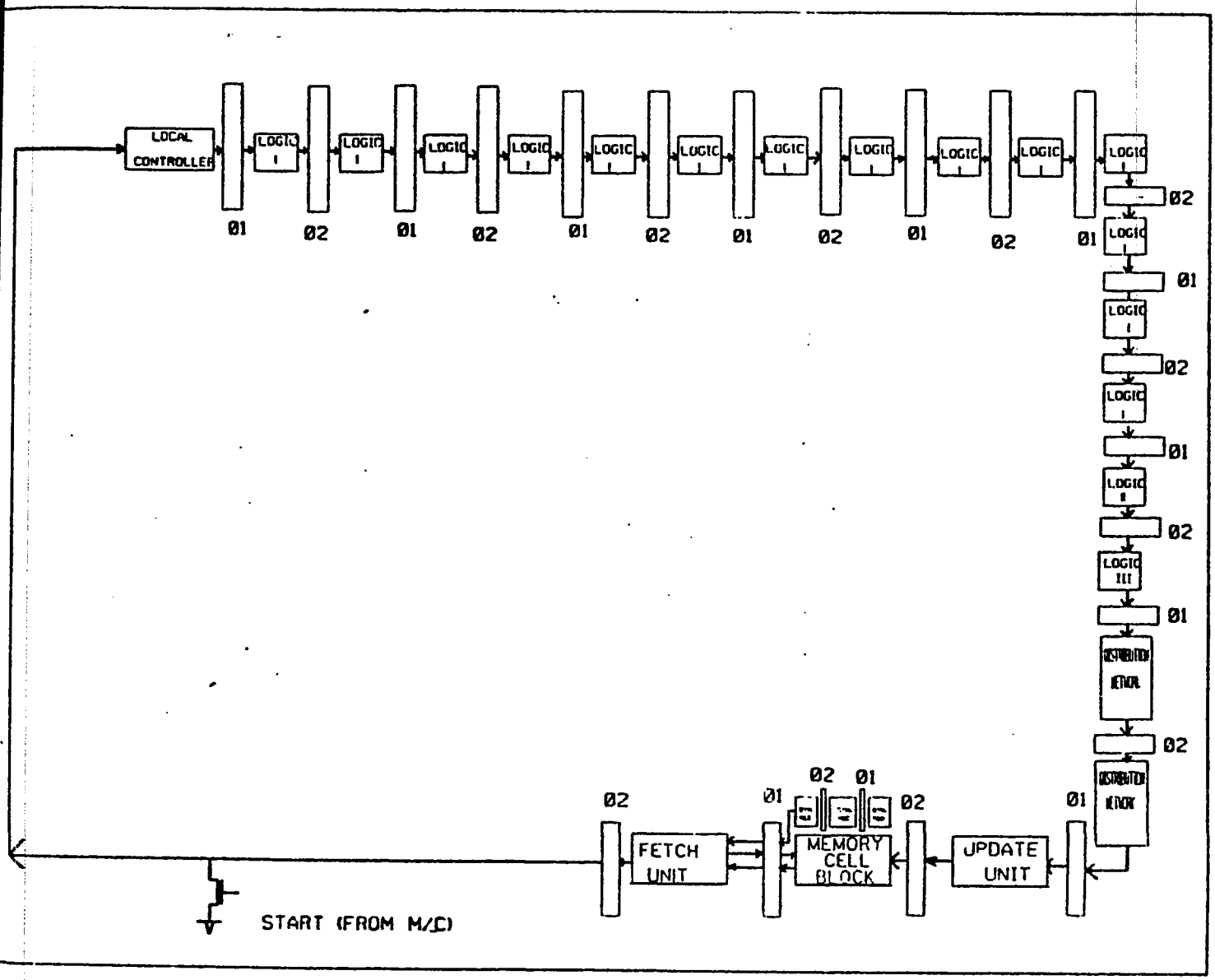


FIGURE 3.4:CIRCULAR PIPELINE ARCHITECTURE

the update unit through the IC address queue and the valid address queue. The fetch unit gives this new instruction to the processing element for computation. This way, the circular pipeline is formed in the architecture.

If all the instruction cells have been computed, then the fetch unit finds a special opcode, the FIN(ish) opcode which tells the fetch unit that there are no more instructions to be executed. The instruction cell containing the FIN opcode has the output of the computation. The FIN opcode is generated by the host computer.

The execution phase of the DFEE system can also be interrupted. One case is, when the update unit writes the result in more than two instruction cells. In this case the IC address queue and the valid address queue are overflown. An update queue overflow signal is given to the master controller. The master controller aborts the operation and raises the error flag to notify the host computer.

The other case is when there is a computation error, .i.e., overflow, divide-by-zero. Computation error is detected by the processing element. The master controller is signaled by the processing element. The master controller raises another error flag to notify the host computer in order to abort the operation resulting from computation error.

Two different error flags are used by the master controller to distinguish the two error conditions (section 4.13).

3.1.1 DATA-FLOW GRAPH EXECUTION ON THE DFEE ARCHITECTURE.

In the data-flow graph represented computation, each result of an instruction forms the operand of another instruction. When the operands reach the instruction, the instruction is enabled for execution.

The result of one instruction has to be routed to different instructions according to the data-flow graph represented computation. Once, all the instructions are executed, a single result is obtained from the data-flow graph.

All the 30 instruction cells in the memory cell block of the DFEE have to be filled by the host computer. Therefore, if the expression results in instructions less than 30 in number, the rest of the instruction cells have to be loaded with blank instructions. The blank instruction consists of the NOOP opcode (0000B), blank operand 1 and operand 2 (100000000B). The Star, Address 1 and Address 2 bits are all zero vectors.

During loading phase, all the instructions are loaded into the memory cell blocks of the DFEE. Depending on the expression, some of the 30 instructions contain constant literals (initial operands with available values). These constant literal instructions are "full" and they are ready to be executed. The termination instruction with the FIN opcode is also present in one of the 30 instructions.

The instructions using the result of the initial full instructions contain blank operands (100000000B), awaiting the execution of the

initial instructions.

Initially, there is no updating and therefore, there is no signal coming from the update unit. The fetch unit selects the full instruction from one of the instruction cells according to the priority of instruction cell IC 1, instruction cell IC 2 (see section 2.3.4). If IC 1 is not full, it sends a blank instruction to the processing element. In the next cycle of execution, if there is still no update signal from the Valid Address Queue, the fetch unit checks for IC 2 and the procedure is repeated. The fetch unit puts an instruction for the processing element only if the instruction is full.

The processing element computes the result and forms the result field. The result field contains the result as well as the destination address of the result. Part of the address is used by the distribution unit to route the result to the cell blocks. The update unit uses the remaining address bits to put the result into one of the instruction cell operand fields, .i.e., either operand 1 or operand 2.

Once the instruction cell has been updated, another instruction is ready to be executed. The fetch unit examines the operands of the updated instruction cell. If the instruction is full, it gives the instruction to the processing element for computation.

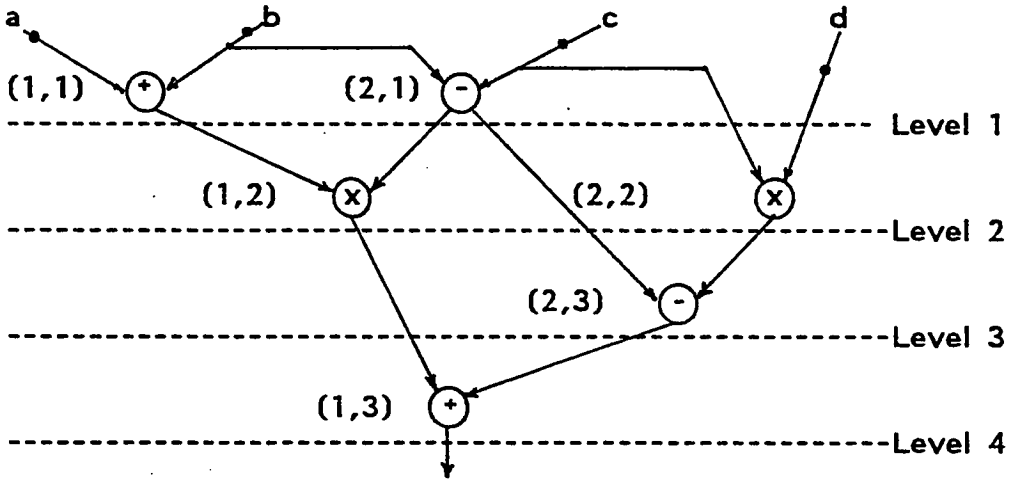
When all the instructions residing in the instruction cell of the memory cell blocks are evaluated in this fashion, the last result is routed to the operand 1 field in one of the instruction cells. This instruction cell contains the FIN opcode.

The fetch unit fetches this instruction cell and upon examination of the opcode FIN, finds that the computation is terminated. The final result of the computation resides in the Operand 1 field of th termination instruction cell. The fetch unit fetches the termination instruction cell, enables the output result to the master controller and raises the valid output flag to notify the master controller that a valid output exists on the output lines.

The master controller enables the output lines to the host computer, resets the DFEE system and waits for more instructions to be loaded into the DFEE system.

Figure 3.5 illustrates the data-flow graph execution on the DFEE architecture.

For the loop schema, the computation to be performed iteratively is done in the same way. However, in this case, a separate data-flow graph computation representing the loop count is also executed. The loop count data-flow graph is merely a counter represented in a data-flow graph. Once the iteration count length is reached, the loop count computation indicates to the computation (which is to be iterated) to stop after the output is received. If the iteration count length is not reached, the output is put back into the initial instruction cell to restart the computation. The two data-flow graphs are shown in Figure 2.9.



a, b, c and d are data-tokens.
($) denotes a blank operand to be filled
XYZ digits in Adr1/2 imply X=Cell #, Y=IC #, Z=Operand #

	CELL BLOCK I						CELL BLOCK II					
IC	# Opcd	Opr1	Opr2	Star	Adr1	Adr2	Opcd	Opr1	Opr2	Star	Adr1	Adr2
1	+	a	b	A	121	000	-	b	c	B	122	231
2	x	($)	($)	A	131	000	x	c	d	A	232	000
3	+	($)	($)	A	251	000	-	($)	($)	A	132	000
4	NOP	($)	($)	A	000	000	NOP	($)	($)	A	000	000
5	NOP	($)	($)	A	000	000	FIN	($)	($)	A	000	000
6	NOP	($)	($)	A	000	000	NOP	($)	($)	A	000	000
	B L A N K N O P's						B L A N K N O P's					
15	NOP	($)	($)	A	000	000	NOP	($)	($)	A	000	000

FIGURE 3.5: ILLUSTRATION OF DATA-FLOW GRAPH EXECUTION ON THE DFEE ARCHITECTURE

3.1.2 CIRCULAR PIPELINE ARCHITECTURE OF THE DFEE

The execution unit of the DFEE system forms a circular pipeline architecture (Figure 3.4). All the sub-units inside the circular architecture have been pipelined. The standard two-phase non-overlapping clocking scheme is used for clocking the circular pipelined architecture.

The processing element is divided into a 18-stage pipeline. The distribution network is divided into two stages. The IC Address Queue and the Valid Address Queue have been divided into four stages. The update unit , the memory cell blocks and the fetch unit have been used as a single stage pipeline.

Each stage inputs at one phase of the two-phase clock and outputs at the other non-overlapping phase of the system clock (phi 1 and phi 2). The longest delay between the logic stages determines the clocking frequency of phi 1 and phi 2 (Figure 3.4). The start signal from the master controller is used to null the sub-units during the loading phase.

A bottom-up design [7] approach is chosen, where the sub-units are designed first and then linked to each other by the circular pipelined data-paths under the control of the master controller.

To describe the hardware in terms of simple description, a PASCAL-like language is used. This approach is chosen for two reasons. Firstly, it describes the hardware in a structured way. Secondly, this description will help in the layout of the DFEE chip,

since a tool for computer aided layout design already exists as BELLE-Basic Embedded Layout Language. This system is embedded in PASCAL [13].

3.2 DISTRIBUTION NETWORK

The function of the distribution network is to route the Result I and Result II received from the Processing Element I and Processing Element II to the correct update unit. The update unit is associated with each cell block. Once the cell block is determined by the distribution network, the distribution network gives the result field to the corresponding update unit. The distribution network discards the cell block (CB field) bit, once it determines to which cell block the result field should go. When the address field is a zero vector, this implies that a null result has come in from the processing element. This happens in the case when the processing element pipeline is being filled and all null results are coming out of the processing element.

3.2.1 GENERAL DESCRIPTION

Figure 3.6 shows the input/output description of the distribution network. The distribution network receives two result fields and outputs four result fields, two result fields to Update Unit I and two result fields to Update Unit II. This is so, since the result from Processing Element I may need to go to Cell Block II and the result from Processing Element II may need to go to Cell Block I.

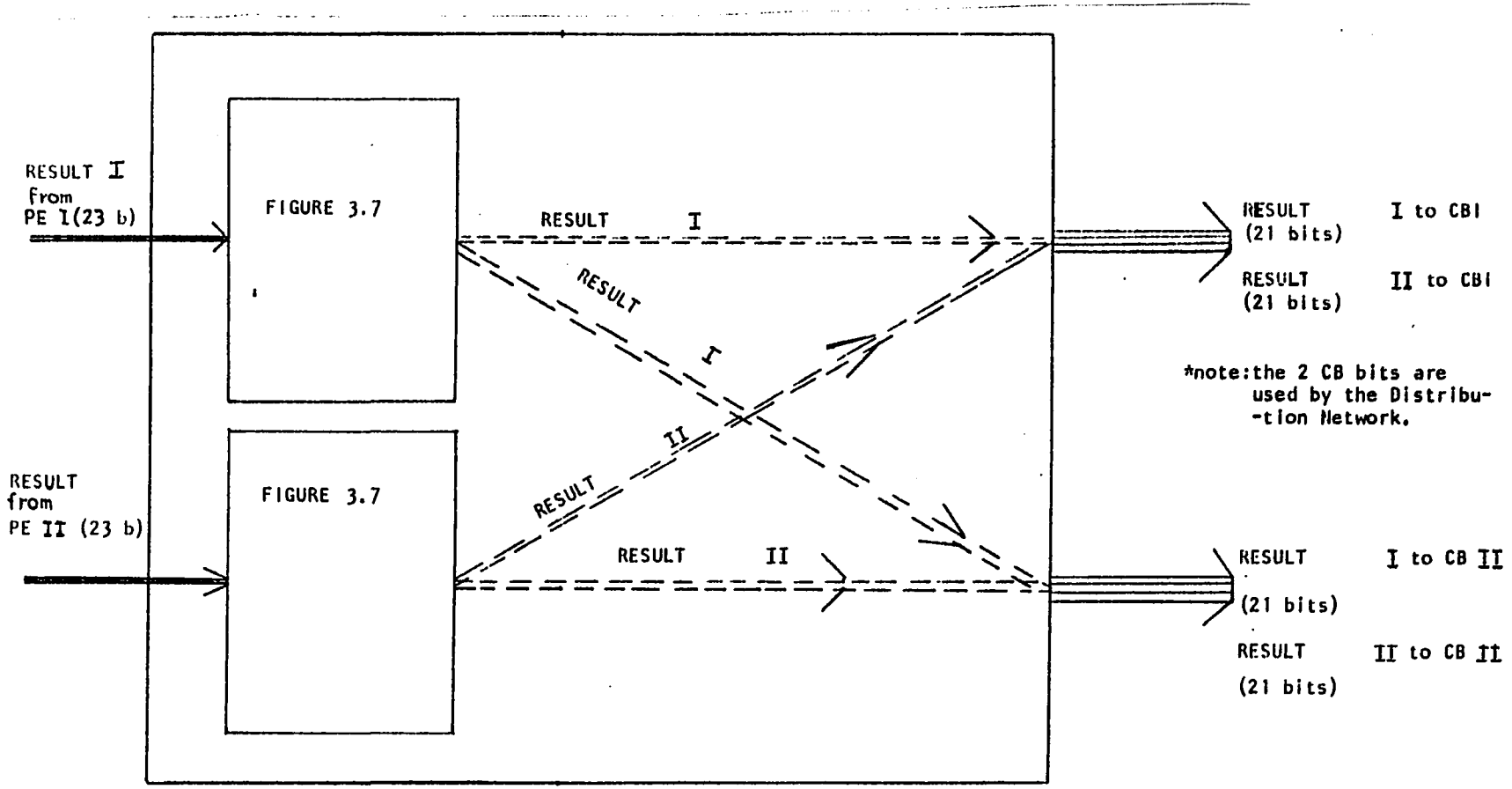


FIGURE 3.6:DISTRIBUTION NETWORK-INPUT/OUTPUT

In the case that only one result field is valid out of the two, the distribution network sets the other result field to an all zero vector.

Figure 3.6 contains two modules detailed in Figure 3.7. Result Field I and Result Field II go through the same logic in parallel. The input to the first stage circuit of Figure 3.7 is the same result with two addresses, Address 1 and Address 2. The output of the third stage circuit of Figure 3.7 is the same result conditionally routed to Cell Block I and Cell Block II. At this stage, the result field contains the Result (9-bit), Address 1 (5-bit), Address 2 (5-bit) and Star (2-bit) sub-fields. The cell block sub-field (CB) has been used by the distribution network.

The first stage circuit of Figure 3.7 is a demultiplexer (DEMUX) with selection inputs as Star (2-bit). Stage two is a demultiplexer with data-selectors as respective cell block (CB-1 bit) values.

Lastly, stage three converges to form the result field. Since the result emerging out of the third stage multiplexer is the same, (14 + 14 =28 from the second stage), the common 9 bits of the result field are discarded from the 28 bits out of the stage three multiplexer. This yields the 21-bit result field as required.

The structure of the DEMUX used in the design is based on the circuit of Figure 3.8. The circuit is for the single bit case and the same structure is used for a 4n-to-n bit DEMUX.

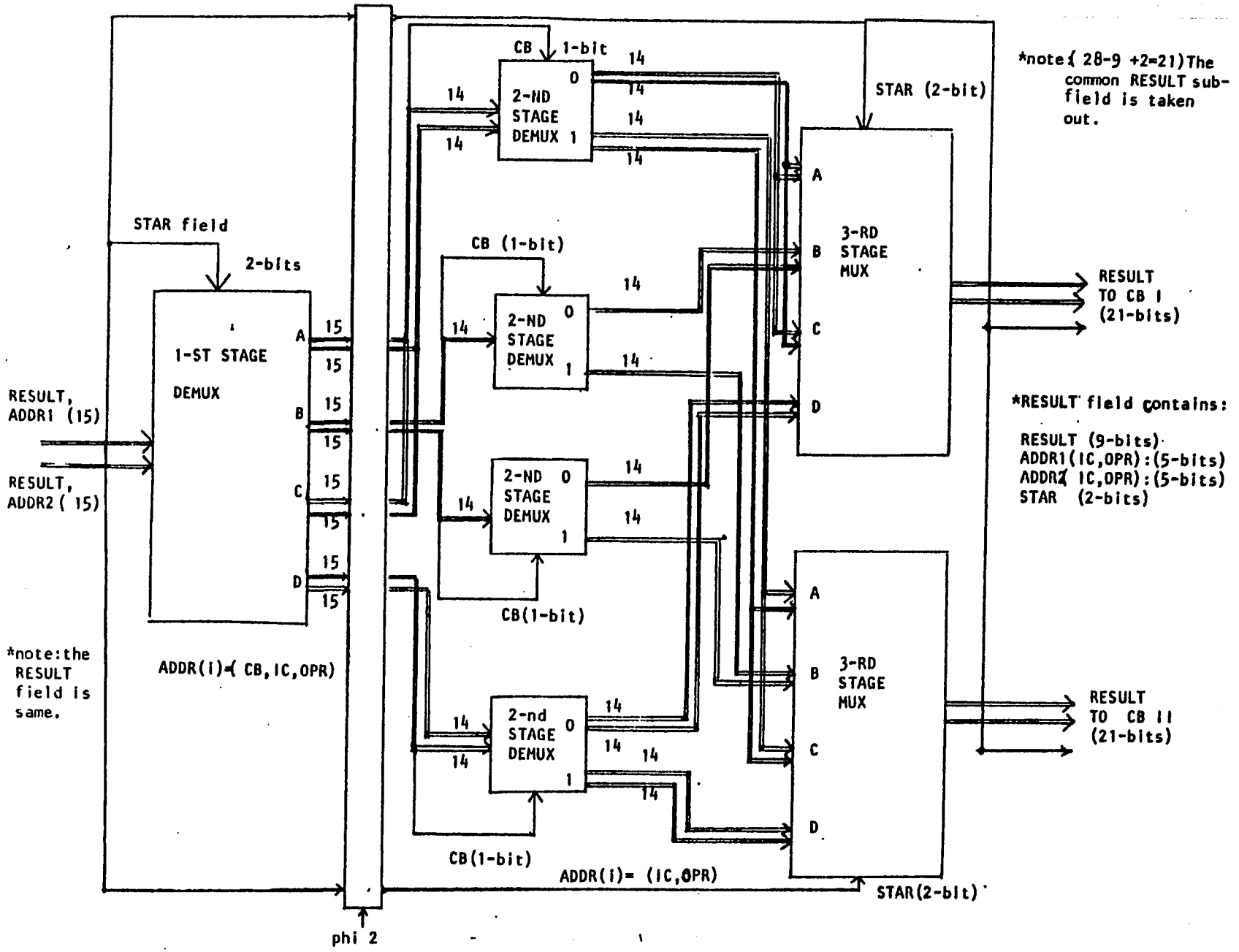

CB 1-bit
2-ND STAGE DEMUX
STAR (2-bit)
*note{ 28-9 +2=21)The common RESULT sub-field is taken out.
STAR field
2-bits
CB (1-bit)
1-ST STAGE DEMUX
3-RD STAGE MUX
RESULT TO CB I (21-bits)
RESULT, ADDR1 (15)
RESULT, ADDR2 (15)
*RESULT field contains:
RESULT (9-bits)
ADDR1(IC,OPR):(5-bits)
ADDR2(IC,OPR):(5-bits)
STAR (2-bits)
CB(1-bit)
*note:the RESULT field is same.
ADDR(i)=(CB,IC,OPR)
2-nd STAGE DEMUX
RESULT TO CB II (21-bits)
CB(1-bit)
ADDR(i)= (IC,OPR)
STAR(2-bit)
phi 2

FIGURE 3.7

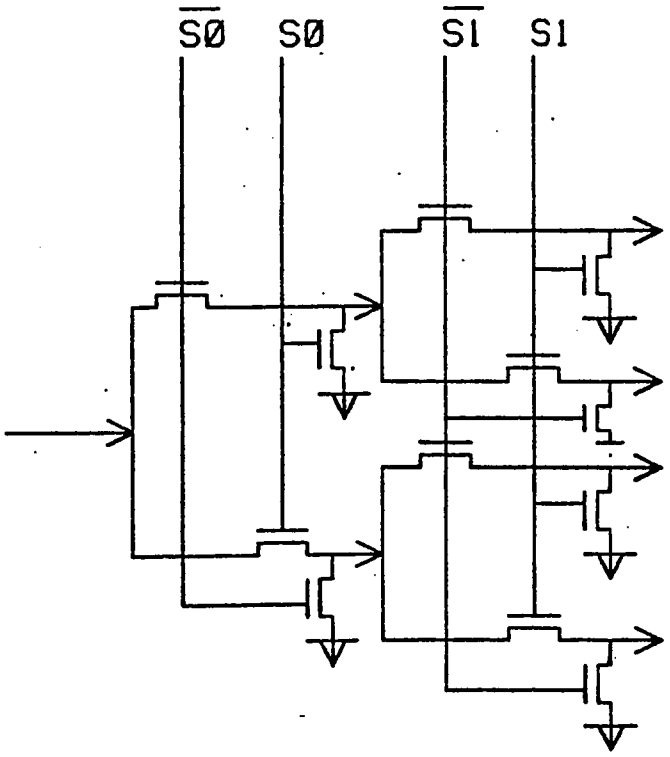


FIGURE 3.8:1-BIT DEMULTIPLEXER WITH 2-BIT CONTROL
(Grounding is used for the distribution network only)

The selection inputs are the same Star bits (2-bits). For the stage two DEMUX, the same structure is repeated; however, instead of using two tree-structured DEMUX, only one is used, since the single bit CB is used as a data-selector. Grounding is used in these structures since, the distribution network conditionally puts an all zero result field to the cell blocks.

The structure of MUX in stage 3 is shown in Figure 3.9. This is for a 4 by 1 MUX with 2 bits of data selection. The structure is extended to nx(4x1) MUX. Grounding is needed in these structures, since the distribution network conditionally sends an all zero result field to the cell blocks.

3.2.2 FORMAL DESCRIPTION

NOTATION:[] represents a field.

e.g[RESULT]:=[RESULT]-[STAR]

denotes that the new result field does not contain the star bits (2-bit).

INPUT:Result Field I from PE I.

Result Field II from PE II.

OUTPUT:[Result Field I , Result Field II] for Cell Block I.

[Result Field I , Result Field II] for Cell Block II.

(**********c o m m e n t**********)

Note:Result Field I and Result Field II contain different

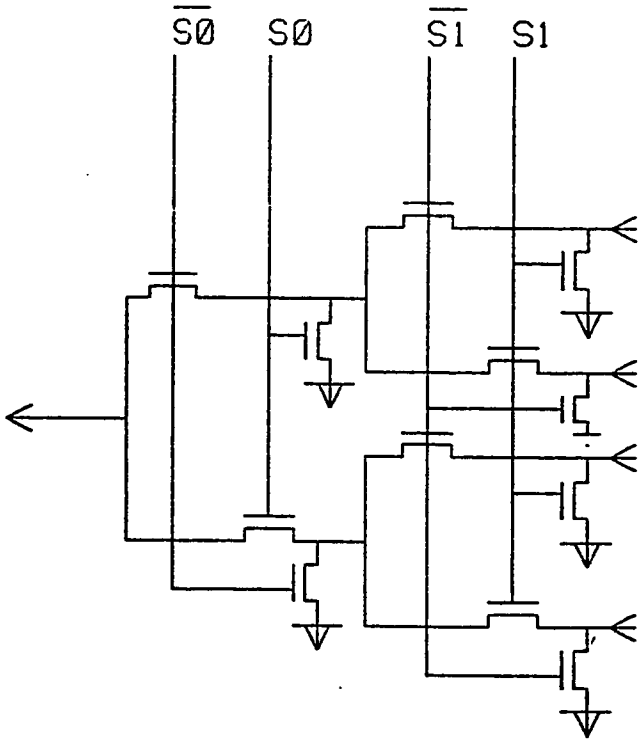


FIGURE 3.9:1-BIT MULTIPLEXER WITH 2-BIT CONTROL
(Grounding is used for the distribution network only)

```
     results (9-bit).
(**************************************************************)

Result Field:=[Result,Star,Addr1(.cb.ic.opr),Addr2(.cb.ic.opr)]
DESCRIPTION OF MODULE FIGURE 3.7

INPUT:Result Field (*either result field I or II*)

OUTPUT:Result Field destined for CB I.
       Result Field destined for CB II.
(*******for first level data-selector*******)
[Result Field]:=[Result Field]-[Star]
(*******i.e, taking out the two Star bits as data-selector****)
(*******[Star]:=[A,B,C,D]*******************)

CASE

If A true then (***1-st stage selection***)
BEGIN
[Addr1]:=[Addr1]-[Addr1.cb]
(***taking out cb value of data selection in stage 2***)
     CASE(***second-stage data-selector***)
         (***Addr1.cb is true when Addr1.cb is logic 1***)
          If not (Addr1.cb) true then
      BEGIN
          Result Field for CB I:=
          [Result,Addr1(.ic.opr),Addr2(.[0000].[0])];
```

```
        Result field for CB II:=[0];
        (*** i.e., a zero vector is put in for the result field
             destined for CB II***)
    END
        If (Addr1.cb) true then
    BEGIN
        Result Field for CB II:=
        [Result,Addr1(.ic.opr),Addr2(.[0000].[0])];
        Result Field for CB I:=[0];
        (*** i.e a zero vector is put in for the result field
             destined for CB I***)
    END
  ENDCASE
END
(*****************c o m m e n t**********************************)
The second stage data-selector puts Address 2 lines to ground
unconditionally.
(*****************************************************************)


If B is true then (***1-st stage data-selector***)
BEGIN
[Addr1]:=[Addr1]-[Addr1.cb];
[Addr2]:=[Addr2]-[Addr2.cb];

(***For Addr1***)
  BEGIN BLOCK 1
    CASE (***2-nd stage selector***)
```

```
      If not (Addr1.cb) true then
    BEGIN
        Partial Result Field 1:=
        [Result,Addr1(.ic.opr)];
        Partial Result Field 2:=[0]
    END
      If (Addr1.cb) true then
    BEGIN
        Partial Result Field 2:=
        [Result,Addr1(.ic.opr)];
        Partial Result Field 1:=[0]
    END

  ENDCASE
END BLOCK 1


(***For Addr2***)
  BEGIN BLOCK 2
    CASE (***2-nd stage selector***)

      If not (Addr2.cb) true then
    BEGIN
        Partial Result Field 1:=
        [Result,Addr2(.ic.opr)];
        Partial Result Field 2:=[0]
    END
      If (Addr2.cb) true then
```

```
        BEGIN
            Partial Result Field 2:=
            [Result,Addr2(.ic.opr)];
            Partial Result Field 1:=[0]
        END

      ENDCASE
    END BLOCK 2

(***merging in stage 3***)
(**********c o m m e n t**********)
The symbol U denotes a union of the two sets.
Physically, this means collectively taking the outputs from
two different sets of lines.
(********Note : 'U' is union of the fields**********)

Result Field for CBI:=[Partial Result Field 1 from Block 1] U
                      [Partial Result Field 1 from Block 2]


Result Field for CBII:=[Partial Result Field 2 from Block 1] U
                       [Partial Result Field 2 from Block 2]

END
If C is true then (***first stage selection***)

      same as the case of A

If D is true then (***first stage selection***)

BEGIN
```

```
[Addr2]:=[Addr2]-[Addr2.cb]
(***taking out cb value of data selection in stage 2***)
    CASE(***second-stage data-selector***)
        (***Addr2.cb is true when addr2.cb is logic-1***)
        If not (Addr2.cb) true then
    BEGIN
        Result Field for CB I:=
        [Result,Addr2(.ic.opr),Addr1(.[0000].[0])];
        Result Field for CB II:=[0];
        (*** i.e., a zero vector is put in for the result field
             destined for CB II***)
    END
        If (Addr2.cb) true then
    BEGIN
        Result Field for CB II:=
        [Result,Addr2(.ic.opr),Addr1(.[0000].[0])];
        Result Field for CB I:=[0];
        (*** i.e., a zero vector is put in for the result field
             destined for CB I***)
    END
   ENDCASE
END
(*****************c o m m e n t**************************************)
The second stage data-selector puts Address 1 lines to ground
unconditionally.
(*****************************************************************)

ENDCASE
```

3.3 UPDATE UNIT

The update unit has the following function:

1) The update unit receives two result fields. Depending on the address of the result fields, the update unit may need to update only one result. The update unit can also copy the same result into two different instruction cell locations. This is achieved by issuing the proper IC address which will enable the write operation into the memory cell blocks. There are two result fields; therefore two such (4-bit) signals are generated by the update unit. These signals are LOAD DEMUX I corresponding to Result Field I and LOAD DEMUX II corresponding to Result Field II. In the case that only one result needs to be updated, one of these signals is given a vector 0000B indicating no update (write) operation in the memory cell blocks.

   In some instances, same result may need to be copied into two different IC locations. For this purpose, a single bit signal, S4 is used which is used in the cell blocks. The purpose of this signal is to make result I the same as result II. S4 is the enabling signal for a pass-transistor, joining the Result I and Result II buses together (see section 3.4).

2) Whenever there is an update operation (i.e., writing the results into the memory cell blocks) performed by the update unit, this has to be communicated to the fetch unit. As the update unit writes the results into the cell block memory, it pushes the address of the IC just updated into the IC address

queue. Further, in order to communicate to the fetch unit that a valid address resides on the IC address queue, the update unit puts a logic-1 value into the valid address queue. This operation is done by enabling controls S1 and S2. Signals LOAD1, LOAD21, LOAD22 are the loading input for the valid address queue. LOADADDR1, LOADADD21, LOADADD22 are the load signals associated with the IC Address Queue.

The input/output configuration of the update unit is shown in Figure 3.10. There are three possibilities

1- In the case that only one result is updated, the IC address is put at the LOADADDR1 and correspondingly the valid address queue is loaded with the true value by LOAD1.

2- In the case that two results are written into the memory cell blocks, both the IC addresses are loaded into the IC address queue by LOADADD21 and LOADADD22. Correspondingly, the valid address queue is loaded with the true values at LOAD21 and LOAD22. This way, the IC address queue and the valid address queue are filled. During the clock phase phi 1 , the fetch unit picks out the rightmost queue value. However, if the next set of results update the cell block in two IC locations, then the previous IC address residing in the IC address queue will be lost. In the case of loosing an address, the update unit does not give any warning.

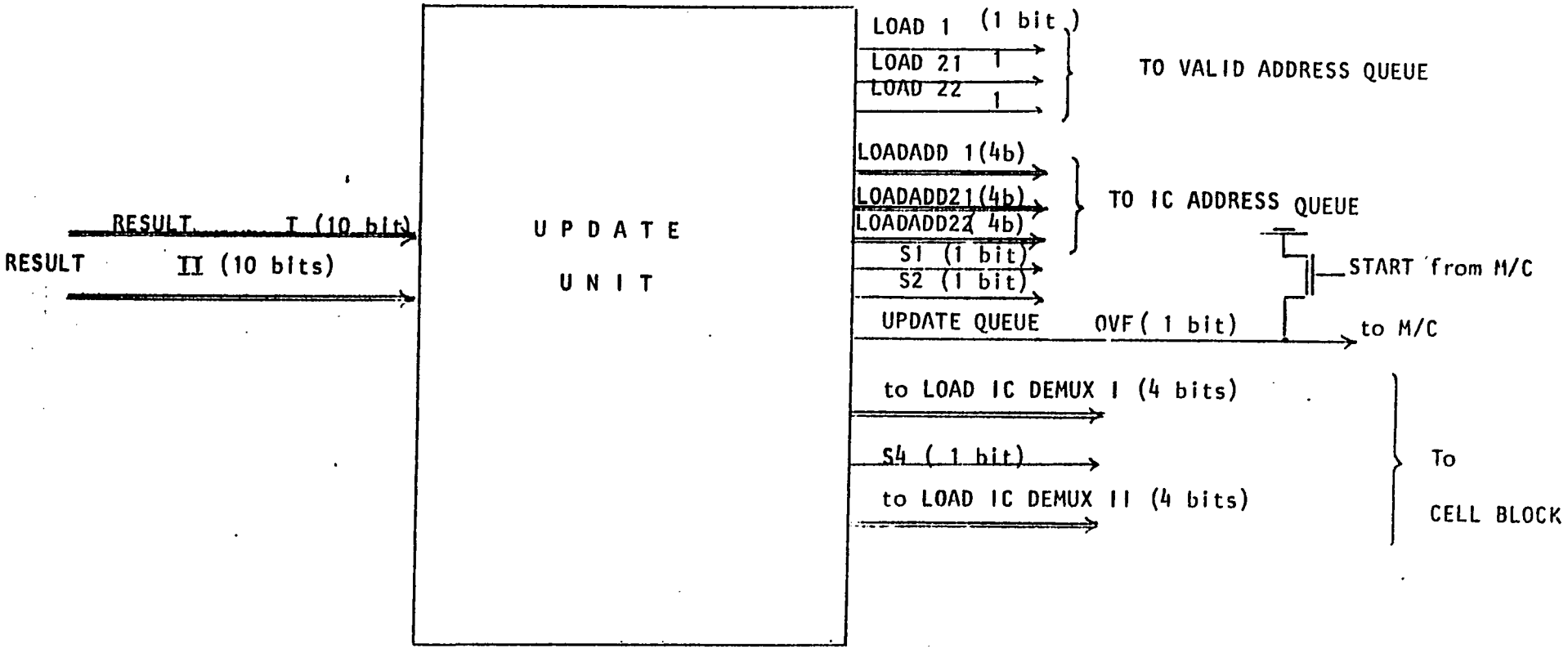


FIGURE 3.10:UPDATE UNIT-INPUT/OUTPUT

The IC Address Queue and the Valid Address Queue are shifted by one bit at each cycle of phi 1 and phi 2.

3- If three updates are done in the same cell block, the update unit gives out the warning to the master controller that the queues have overflown.

The Valid Address Queue and the IC Address Queue are shown in Figure 3.11 and Figure 3.12 respectively. The figures show the implementation of the control and shift operations mentioned above. The queues are set to zero at the START signal from the master controller. Dynamic shift registers are used with phi 2 as the loading phase and phi 1 as the output phase to the fetch unit.

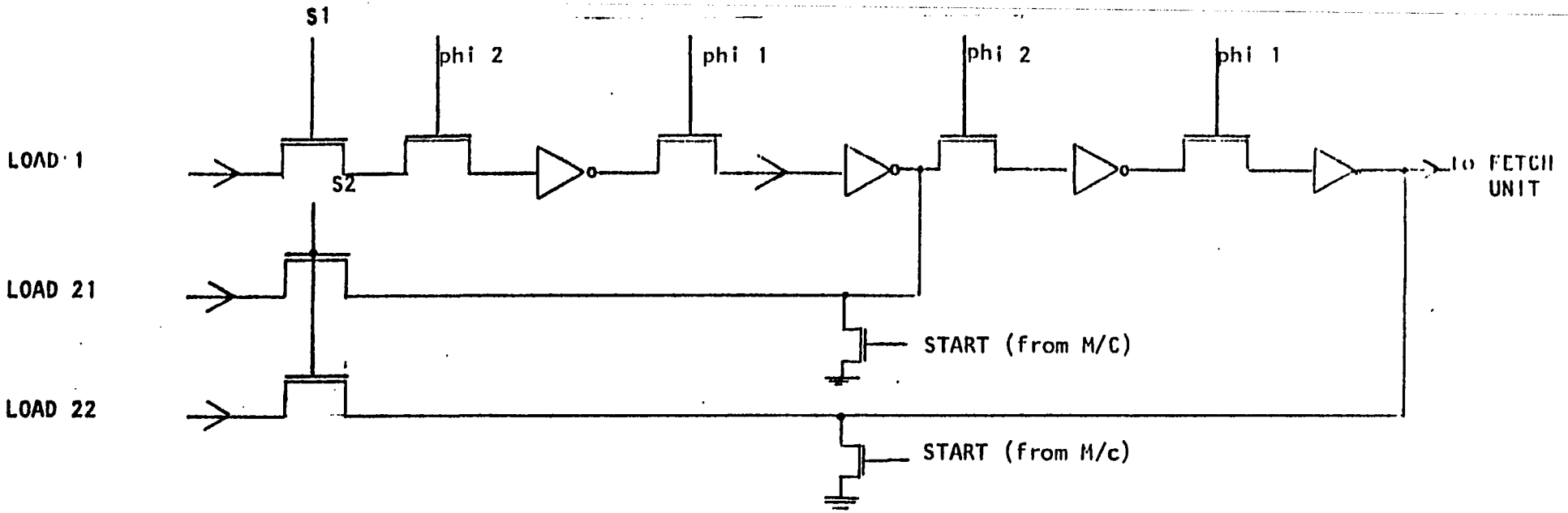
3.3.1 NOTATION OF ADDRESS REPRESENTATION

There are two addresses belonging to the two result fields, Address I and Address II. Each of these addresses consist of two addresses; Address 1 and Address 2. Address 1 and Address 2 consist of two sub-fields; the IC address (4-bit) and the OPR address (1-bit).

In order to represent any of these bits separately, the following notation is followed:

ADDR I.ADDR 1.IC

This represents the address of the (4-bit) instruction cell address of Address 1 belonging to Address I. Address I is originally contained in Result Field I.



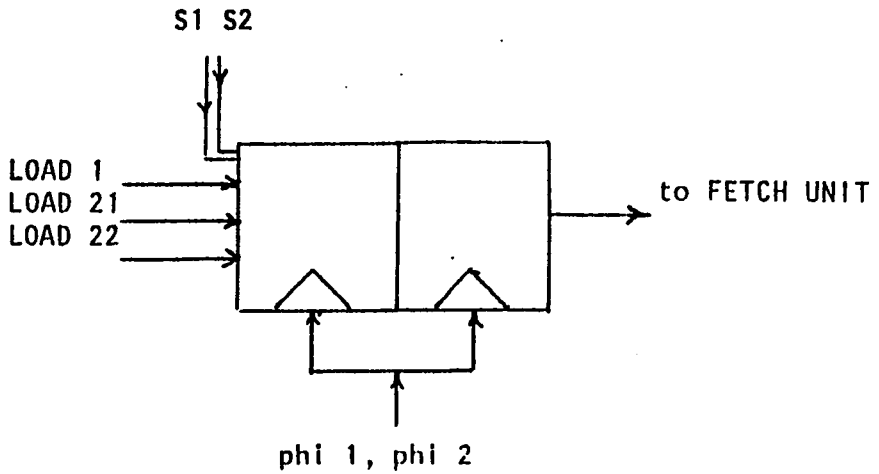


FIGURE 3.11: VALID ADDRESS QUEUE

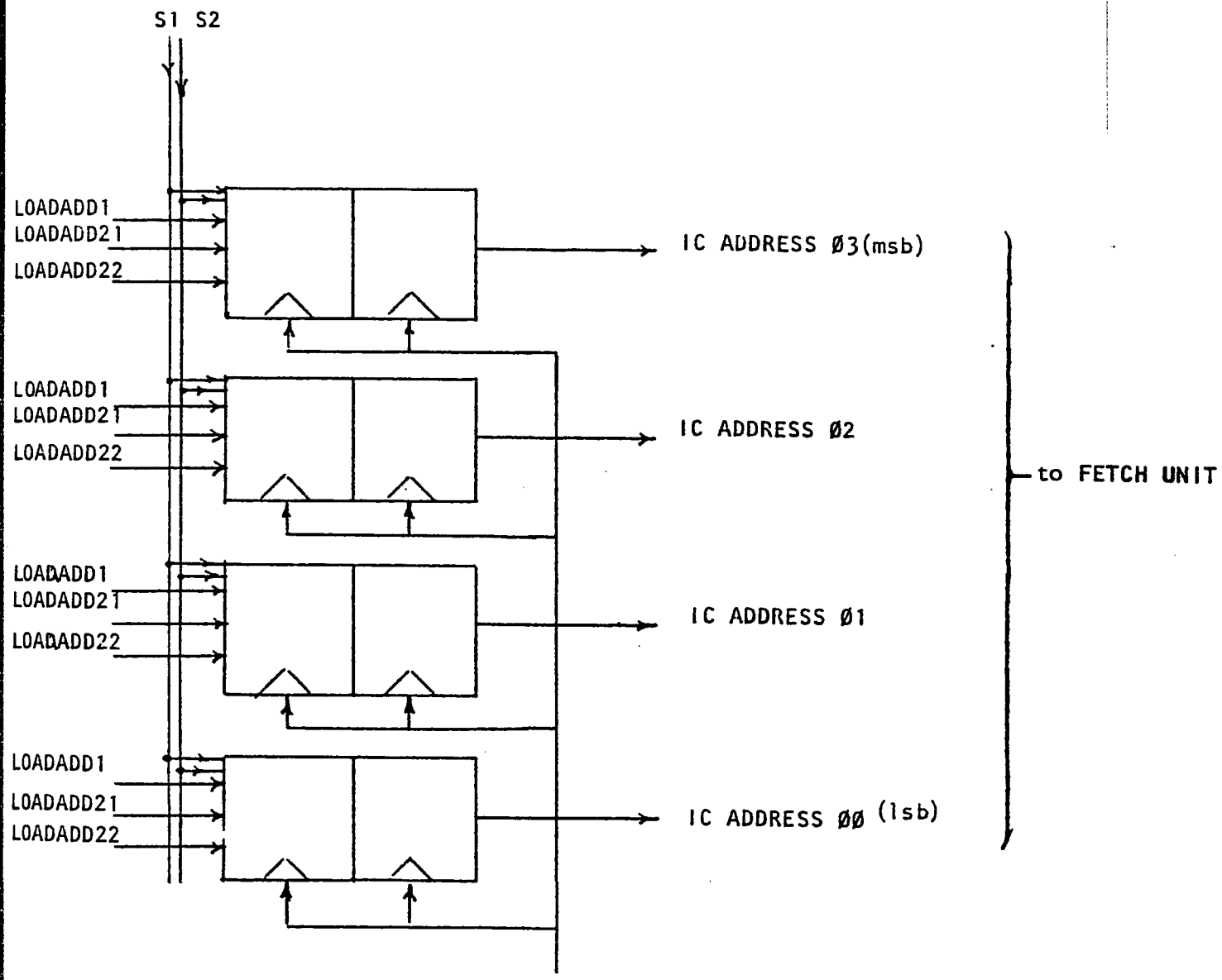


FIGURE 3.12:IC ADDRESS QUEUE

3.3.2 HARDWARE IMPLEMENTATION OF THE UPDATE UNIT

The update unit receives two result fields (Result Field I and Result Field II). The result field coming from the distribution network consists of the following subfields

1) Result (9-bits)
2) Address 1 and Address 2 (total of 10 bits since address.CB is used by the distribution network).
3) Star (2-bits to choose the valid address between address 1 and address 2).

Figure 3.13 shows the input into the update unit demultiplexer logic. Result I and Result II sub fields are connected directly to the cell block Result I and Result II bus. These buses are 9-bits each. Only IC part of Address 1 and Address 2 is input into the update unit. This is done because the IC address will enable the write operation into the cell block for writing the result in either Operand 1 or Operand 2.

The address.opr determines whether the result should be written in Operand 1 or Operand 2. The address.opr is transmitted directly to the cell block. The Star I and Star II bits corresponding to Result Field I and Result Field II are used as the input to the data selection operation for the update unit. Star I and Star II are two bits each and therefore sixteen possible enable signals are generated out of the update demux logic. ADDRI.ADDR1.IC, ADDRI.ADDR2.IC, ADDRII.ADDR1.IC, ADDRII.ADDR2.IC, are inputted into the demultiplexer of Figure 3.13.

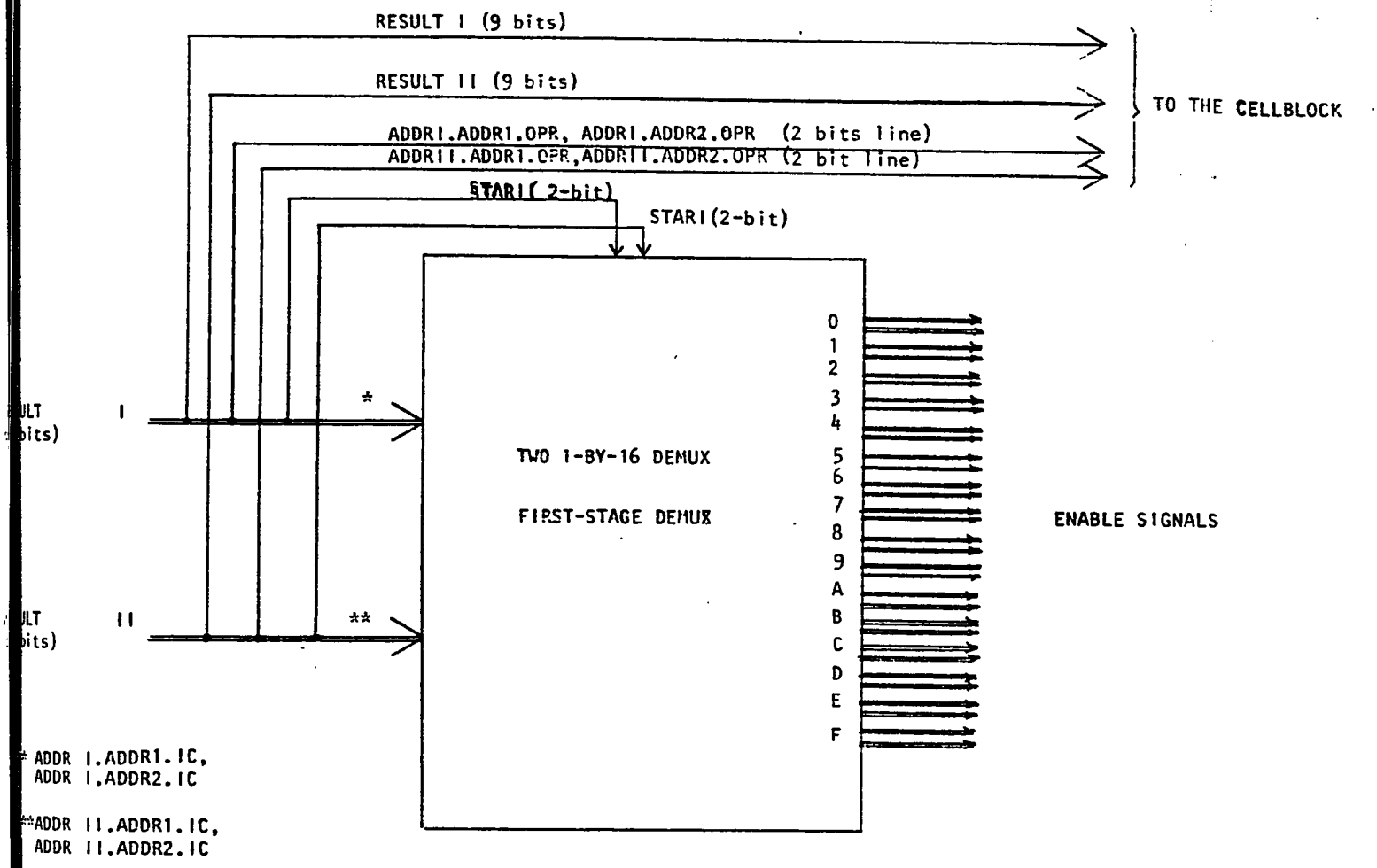


FIGURE 3.13:ENABLE SIGNAL GENERATION

Depending on one of the sixteen enable signals from the update demultiplexer, the two IC address are used to write the two results into the memory cell block. Further, the two IC addresses are loaded into the IC Address Queue for the fetch unit. The Valid Address Queue is put to true value to indicate to the fetch unit that the IC address queue contains a valid IC address.

In the case that more than two instruction cells are written in by the update unit, then this condition is for UPDATE QUEUE OVERFLOW (UPDATE QUEUE OVF). The update unit generates this 1-bit error signal to the master controller.

3.3.3 OUTPUT GENERATION

The update unit generates the output (Figure 3.10) in the following manner:

There are 16 enable signals generated from the circuit of Figure 3.13. Each enable signal activates a logic. This logic inspects ADDRI.ADDR1.IC, ADDRI.ADDR2.IC, ADDRII.ADDR1.IC, ADDRII.ADDR2.IC, for a zero vector (4-bit) when all the four bits of the instruction cell address are zero.

The Star I and Star II bits convey the information as to which of the four instruction cell addresses should be inspected. This information is given as one-out-of 16 enable signals. For each Star I and Star II combination, there is an enable signal (Figure 3.13).

Star I and Star II fields give rise to 16 possibilities

AA,AB,AC,AD,BA,BB,BC,BD,CA,CB,CC,CD,DA,DB,DC,DD

where A=00, B=01, C=10, D=11.

Consider the case when Star I = A and Star II = A. A implies that ADDR1.IC is the valid destination instruction cell address. In this case, the enable signals activate the logic which gives four possible combinations. The symbol "'" denotes a NOT boolean operation while the symbol "." denotes a boolean AND operation.

1)[ADDRI.ADDR1.IC]'.[ADDRII.ADDR1.IC]'

Meaning:All the four bits of ADDRI.ADDR1.IC and of ADDRII.ADDR1.IC are zero.

2)[ADDRI.ADDR1.IC]'.[ADDRII.ADDR1.IC]

Meaning:All the four bits of ADDRI.ADDR1.IC are zero and not all the four bits of ADDRII.ADDR1.IC are zero.

3)[ADDRI.ADDR1.IC].[ADDRII.ADDR1.IC]'

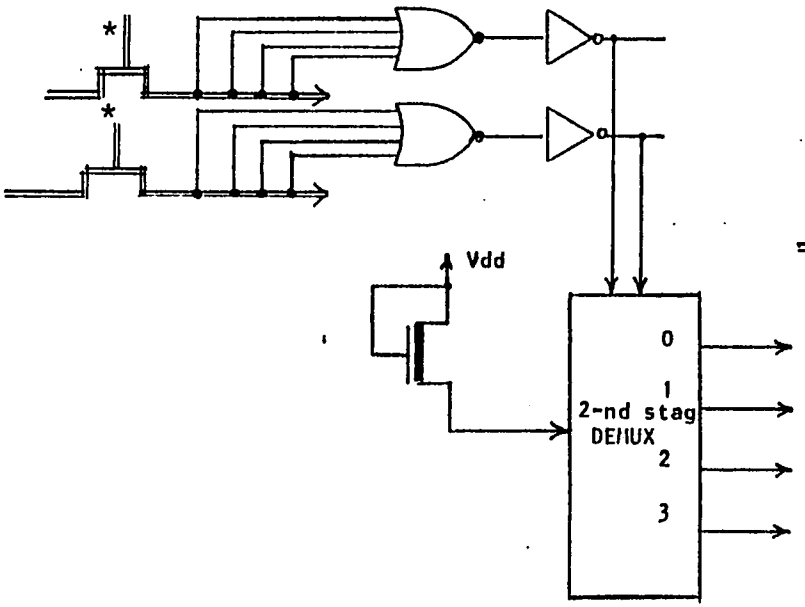
Meaning:Not all the four bits of ADDRI.ADDR1.IC are zero and all the four bits of ADDRII.ADDR1.IC are zero.

4)[ADDRI.ADDR1.IC].[ADDRII.ADDR1.IC]

Meaning:Not all the four bits of ADDRI.ADDR1.IC and of ADDRII.ADDR1.IC are zero.

The four bit IC address field is converted to a single bit for distinguishing the all zero (4-bit) pattern.

Only one out of these possibilities produces the output defined in Figure 3.10. The logic shown in Figure 3.14 generates these two bits from ADDRI.ADDR1.IC and ADDRII.ADDR1.IC lines.



* enable signal from FIGURE 3.13

FIGURE 3.14:2-BIT SELECTION LOGIC

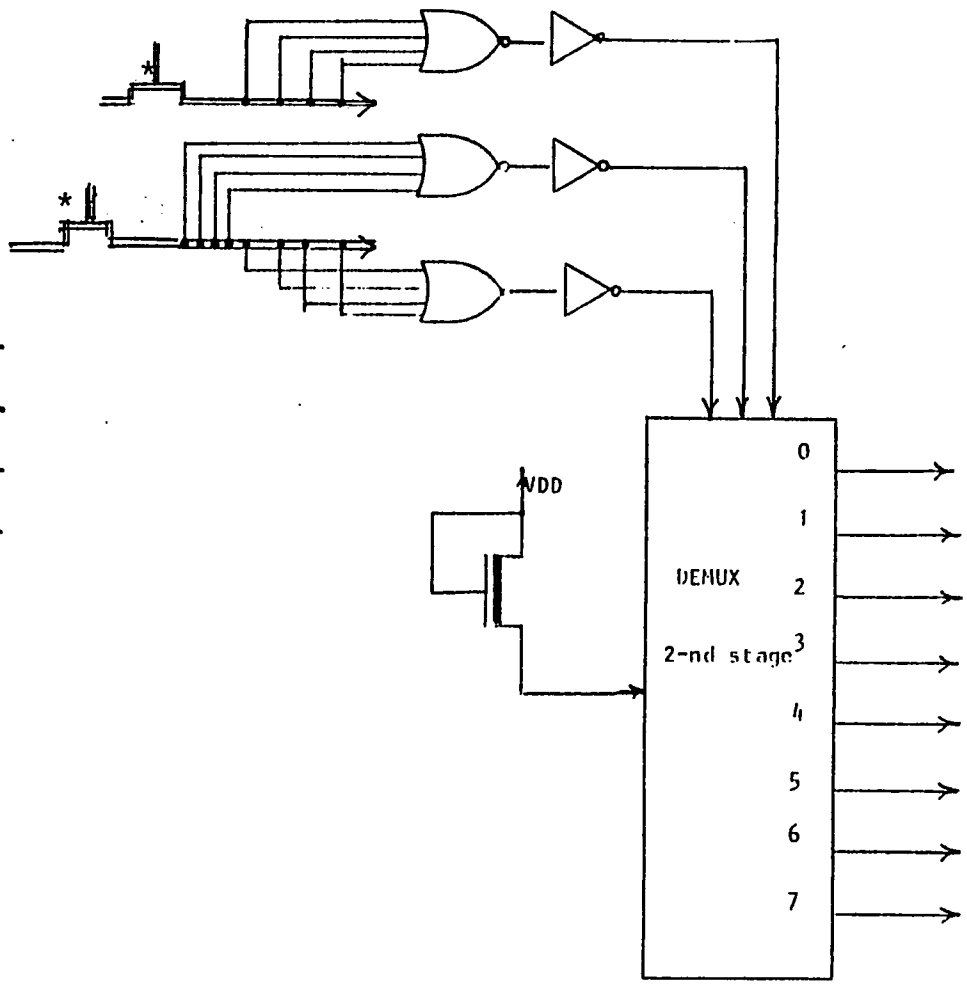


FIGURE 3.15:3-BIT SELECTION LOGIC

The logic checks for the zero vector and operates the two-bit data-selector demultiplexer (DEMUX) as shown in Figure 3.14. The DEMUX generates the four required enable signals only one out of which is active high.

Figure 3.15 and Figure 3.16 transform the other instruction cell address possibilities into 3-bit and 4-bit selection lines to be used by the DEMUX. The DEMUX is also shown in Figure 3.15 and 3.16.

The output of these demultiplexers shown in Figures 3.14 to 3.16, enable the constant charge lines. These constant charge lines are permanently connected to either ground (GND) or logic-1 (VDD). The description of these line values is given under Appendix F (The description of output generation). The line values are structured under BEGIN and END blocks.

3.4 MEMORY UNIT

The memory of the DFEE consists of the instruction cells. The instruction cells store the program representing the data-flow computation.

In this DFEE architecture, the memory unit is restricted to:

1) Two memory cell blocks.
2) Fifteen instruction cells per cell block.

3.4.1 REQUIREMENTS ON THE MEMORY BY THE DFEE ARCHITECTURE

i) All the instruction cells of cell blocks I and II should be connected to form a one directional stack of length 30. The

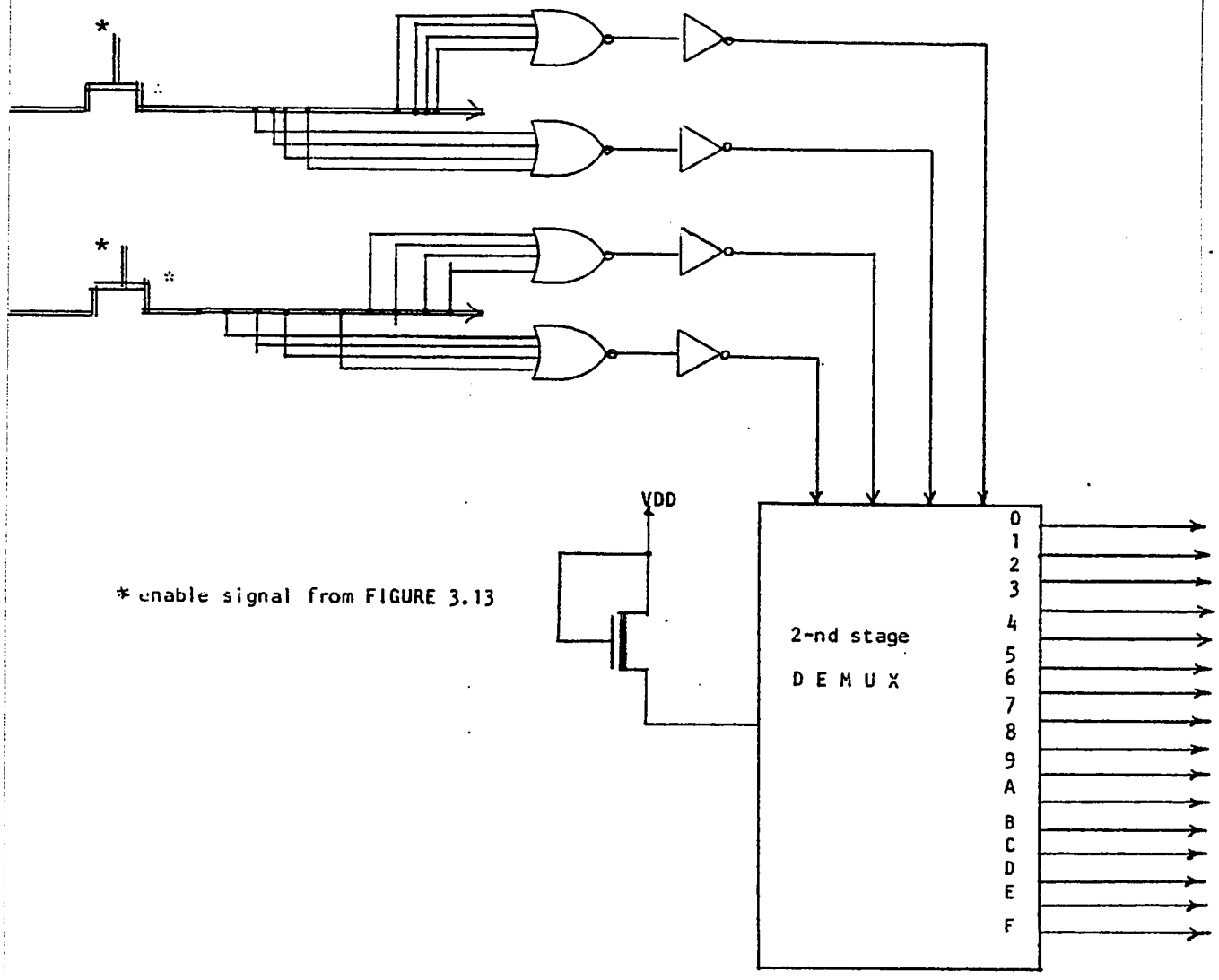


FIGURE 3.16:4-BIT SELECTION LOGIC

stack connection should only be valid in the loading phase of the DFEE. The loading of the instructions into the instruction cells should be under the signal LOAD from the master controller. The memory unit should be designed as an assemblage of shift registers with parallel input and output facility.

ii) Since the update unit can update two results in the memory at the same time, provisions should be made to do so. Two buses, RESULT I and RESULT II load the results into the writable fields of the instruction cell. The writable fields of the instruction cells consist of Operand 1 and Operand 2.

iii) The write operation in the memory should be done during phi 2 while the read operation done by the fetch unit should be during phi 1.

3.4.2 DESIGN OF MEMORY CELL BLOCK

The instruction cell bits are divided into three parts:

i) Instruction cell bits which are only read by the fetch unit and written only at the time of loading. This cell type for a single bit is shown in Figure 3.17.

ii) The instruction cell bits which are written by the update unit into operand 1. This is shown as TYPE I in the instruction cell format in Figure 3.18. The realization of this cell-type for a single bit is shown in Figure 3.19. In order to write Operand 1, write control is ANDed with the 1-bit complemented address of the operand (ADDR.OPR=logic 0).

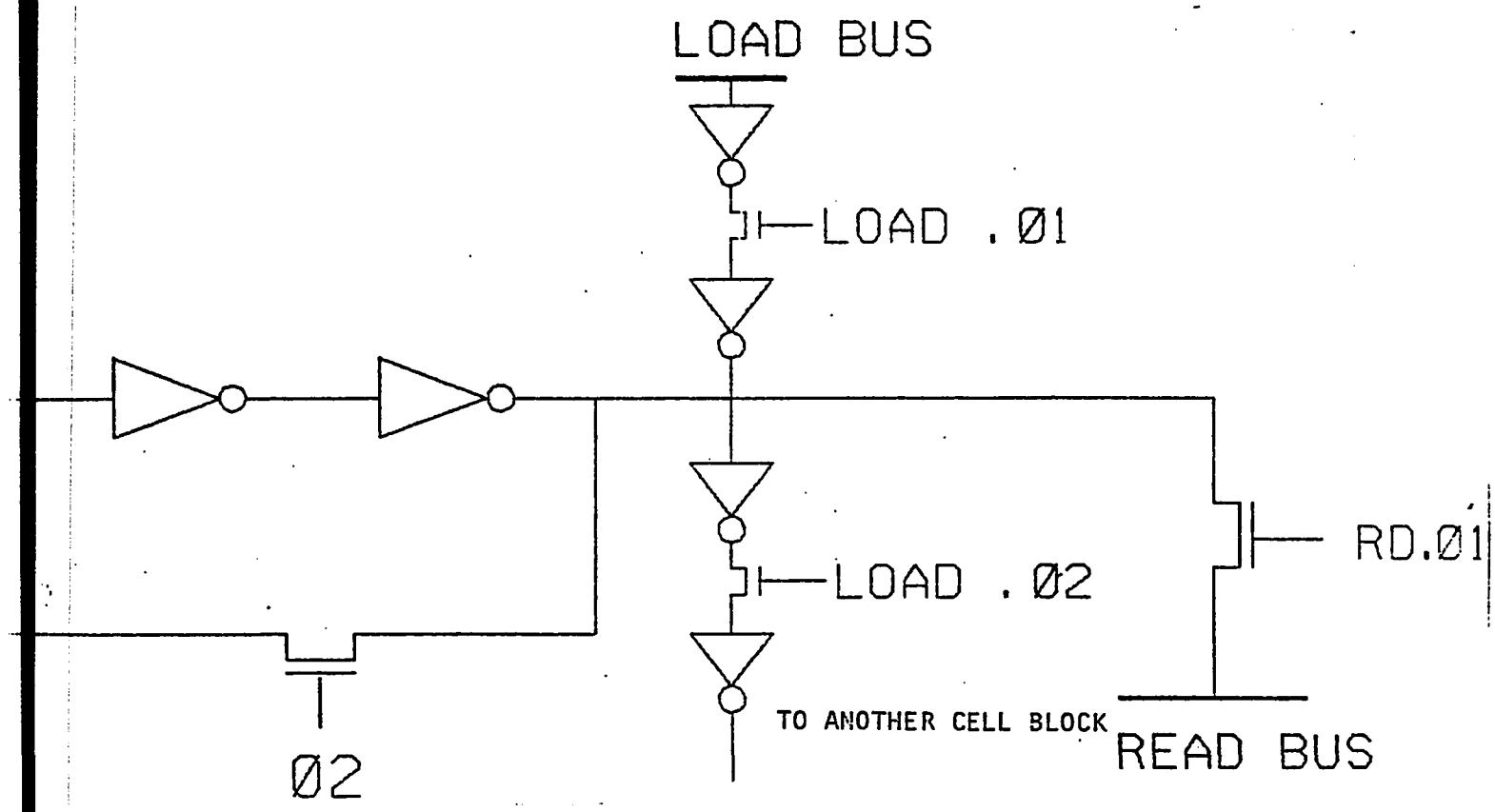

LOAD BUS
LOAD .Ø1
LOAD .Ø2
RD.Ø1
TO ANOTHER CELL BLOCK
READ BUS
Ø2

FIGURE 3.17

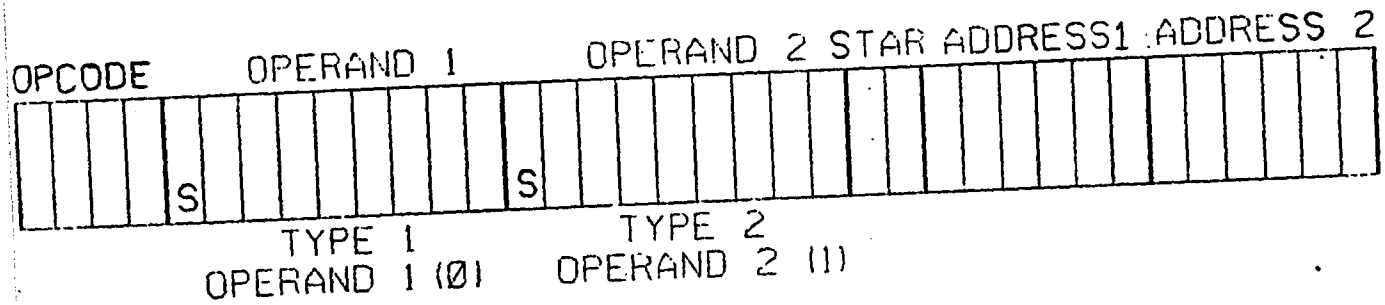


FIGURE 3.18:FORMAT OF THE INSTRUCTION CELL

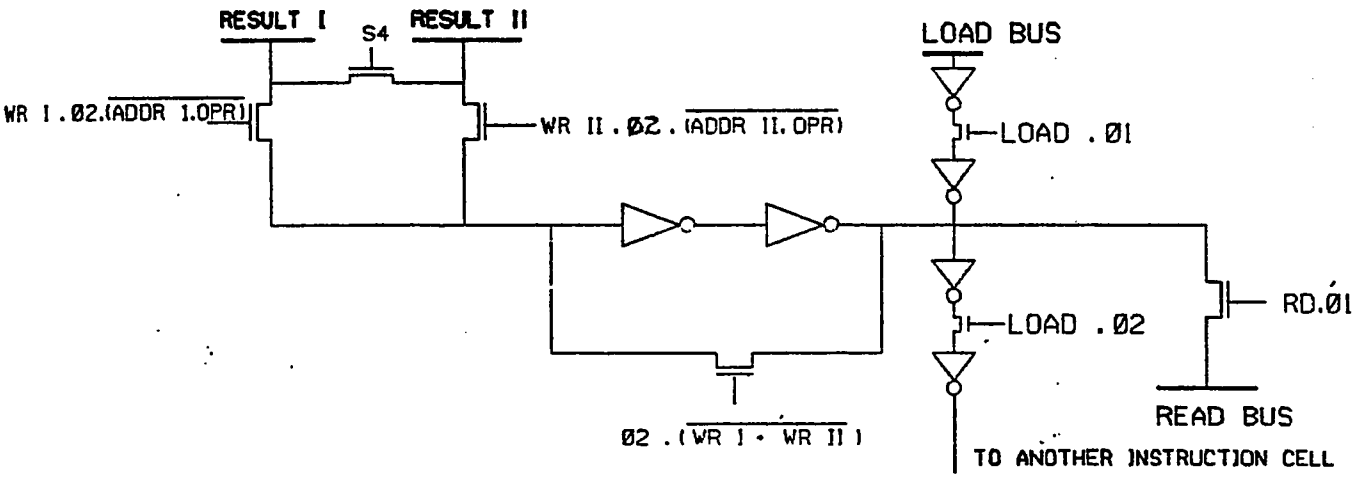

RESULT I
S4
RESULT II
LOAD BUS
WR I . Ø2.(ADDR I.OPR)
WR II . ØZ . (ADDR II. OPR)
LOAD . Ø1
LOAD . Ø2
RD.Ø1
READ BUS
Ø2 . (WR I + WR II)
TO ANOTHER INSTRUCTION CELL

FIGURE 3.19

iii) The instruction cell bits which are written by the update unit into operand 2. This is shown as TYPE II in the instruction cell format in Figure 3.18. The realization of this cell-type for a single bit is shown in Figure 3.20. The write control is ANDed with the uncomplemented single bit address of the operand (ADDR.OPR = logic 1).

Figure 3.21 shows the complete design of one cell block unit. There is one load bus. The two buses RESULT I and RESULT II are shown separately, since they carry different data. In the case that the update unit requires to write the same result in two different instruction cells, pass-transistor S4 is turned on by the update unit (Figure 3.19, 3.20)

In Figure 3.21, ADDRI.OPR and ADDRII.OPR lines are shown coming into the cell block to fill Operand 1 or Operand 2 cell. This is because any result can either be destined for Operand 1 or Operand 2. ADDRI.OPR corresponds to RESULT I and ADDRII.OPR corresponds to RESULT II.

The write control signals are generated by the WRITE DEMUX I and WRITE DEMUX II. The inputs to these demultiplexers are LOAD IC DEMUX I and LOAD IC DEMUX II. The LOAD IC DEMUX signal comes from the update unit corresponding to the cell block. If the LOAD IC DEMUX is 0000B, then no data is written into the IC because this address does not exist (Figure 3.21). Each IC can be filled with one of the two data; RESULT I and RESULT II; therefore, two demultiplexers must be used in the design.

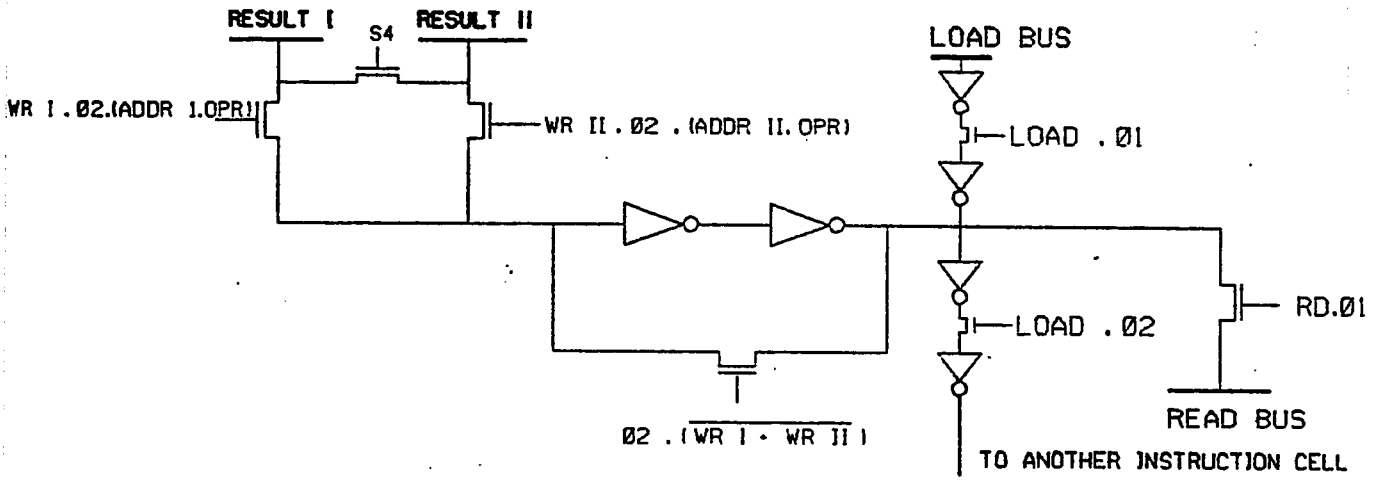

RESULT I
S4
RESULT II
LOAD BUS
WR I . Ø2.(ADDR 1.OPR)
WR II . Ø2 . (ADDR II. OPR)
LOAD . Ø1
LOAD . Ø2
RD.Ø1
READ BUS
Ø2 . (WR I + WR II)
TO ANOTHER INSTRUCTION CELL

FIGURE 3.20

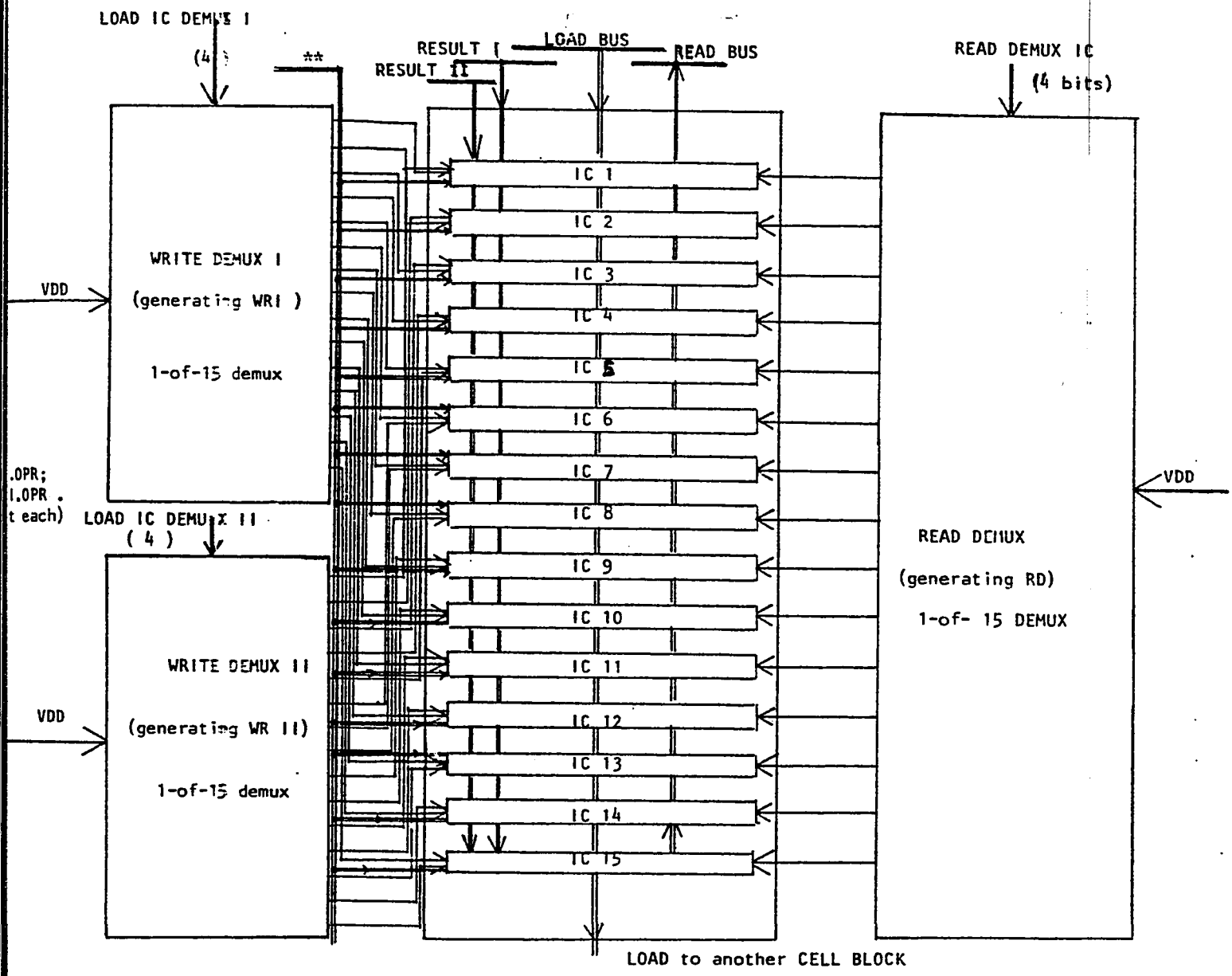


FIGURE 3.21:CELL BLOCK UNIT

The fetch unit associated with the cell block only reads one IC during one phase. Therefore, only one READ DEMUX is used. The read control signal is given as a 4-bit signal READ DEMUX IC from the fetch unit. Similar to the WRITE DEMUX, if the selected IC address is 0000, no IC is read from the memory. The READ BUS does not contain any valid data.

3.5 FETCH UNIT

The purpose of the fetch unit is to fetch the instructions residing in the instruction cell of the memory cell block. If the fetched instruction cell is full, the fetch unit gives the instruction to the processing element.

The fetch unit takes the Valid Address Queue signal with priority. If the Valid Address Queue signal is not true (i.e., logic-0) then the fetch unit starts fetching the full instruction cells in the order of priority IC1, IC2, IC3. Once the fetch unit has checked an IC to be full, it does not check the same IC again unless the update unit gives the IC address to the fetch unit through the IC address queue.

The fetch unit has been designed so that it will not fetch the same IC twice in a row. A local 4-bit register has been included inside the fetch unit for this purpose. This register contains the previous value of the fetched IC address.

The fetch unit uses a local modulo-15 up counter. The counter is incremented by one if the fetched IC address is not from the IC

address queue and the fetched IC is full. This way, the fetch unit keeps track of how many instruction cells have been checked. The counter is set to one from the master controller at the start of the execution phase.

3.5.1 ALGORITHMIC STATE MACHINE DESCRIPTION OF THE FETCH UNIT

FETCHUNIT Input: Valid address flag, IC address from IC queue

Output: Valid output flag, instruction field transfer to the processing element, valid output transfer to the master controller;

LOCAL WORKING UNITS: 4-bit register, 4-bit counter.

1. IF valid address flag on THEN

a.Take the IC address from the valid address queue.

b.Fetch the instruction from the memory cell block.

IF FIN opcode found in the fetched IC THEN

a. Put valid output flag on for the master controller.

b. Put operand 1 on the output bus for the master controller.

ELSE

IF fetched IC is single operand instruction THEN

IF Operand 1 is not blank AND fetch IC value is not equal to the register value THEN

a.Put instruction onto the bus for the processing element.

b.Register:= IC address just fetched.

c. QUIT.

ELSE GO TO 2.

ELSE(*double operand*)

IF operand 1 and operand 2 not blank AND fetched IC value is not equal to register value THEN

a.Put instruction onto the bus for the processing element.

b.Register:=IC address just fetched.

c. QUIT.

ELSE GO TO 2.

ELSE GO TO 2(*for step 1*).

2. IF counter value is greater than number of ICs THEN

a.Put blank instruction on the processing element bus.

b. QUIT.

ELSE

a.Get the IC address from the counter value.

b.Get the instruction from the memory cell block.

c.Counter:=Counter+1;

IF fetched IC is single operand instruction THEN

IF operand 1 is not blank and fetched IC value is not equal to the register value THEN

a.Put instruction onto the bus for the processing element.

b.Register:=IC address just fetched.

c. QUIT.

ELSE GO TO 3.

ELSE(*double operand*)

IF operand 1 and operand 2 are not blank and fetched IC value is not equal to the register value THEN

a.Put instruction onto the bus for the processing element.

b.Register:=IC address just fetched.

c. QUIT.

ELSE GO TO 3.

3.Send blank instruction onto the bus for the processing element.

4. QUIT

END FETCHUNIT.

3.5.2 DESCRIPTION OF LOCAL COUNTER IN THE FETCH UNIT

The counter in the fetch unit is used for obtaining the IC address once the valid address signal is received as false. The requirements on the counter are:

* It should be loadable to one with the start signal from the master controller. It will remain at one and will not count unless the start signal goes low. Once, the counter is set to one by the start signal, the start signal should go low in order to operate the counter .
* The counter should be enabled by putting the start signal to low. The clock-in should be active high in order for the counter to count. This is shown in Figure 3.22 where the clock-in is ORed with the initial start signal to load the counter to one. The counter is disabled and retains its count once the clock-in is low.
* Once the counter enters 1111B, the highest count, it should enter into 0000B state in the next phase and remain there. This is shown in Figure 3.23. This state will indicate to the fetch unit that the count value has exceeded the number of ICs.

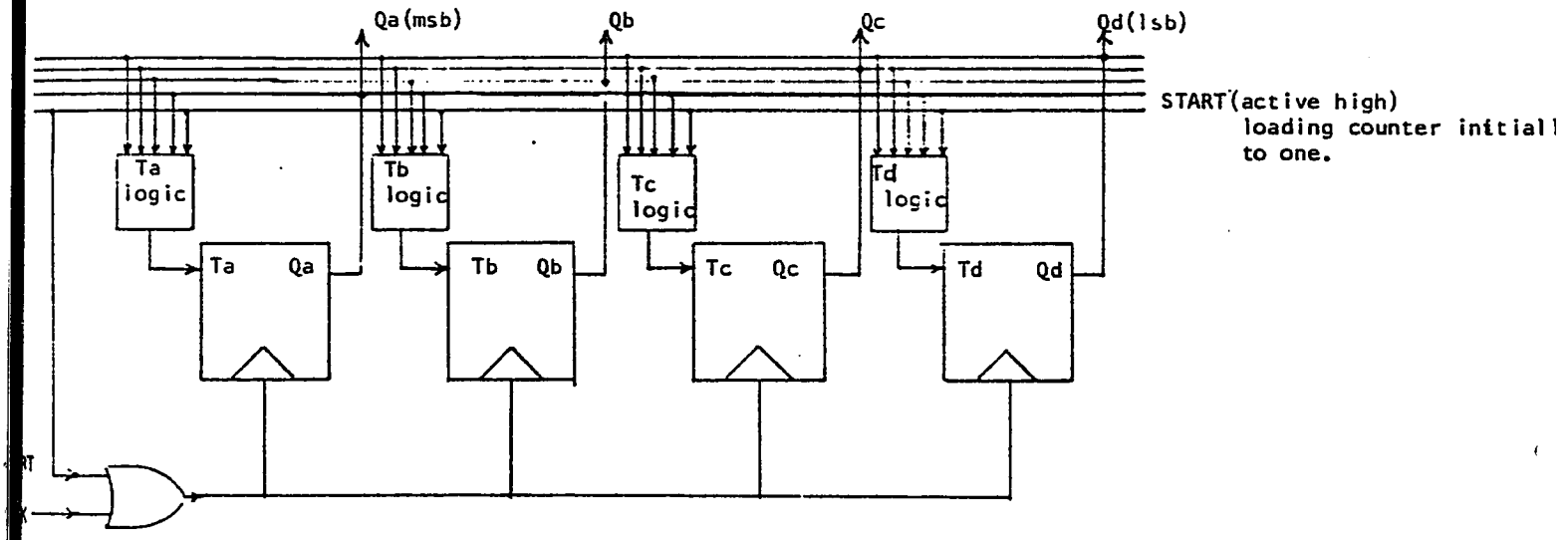


FIGURE 3.22:LOCAL 4-BIT COUNTER(static)

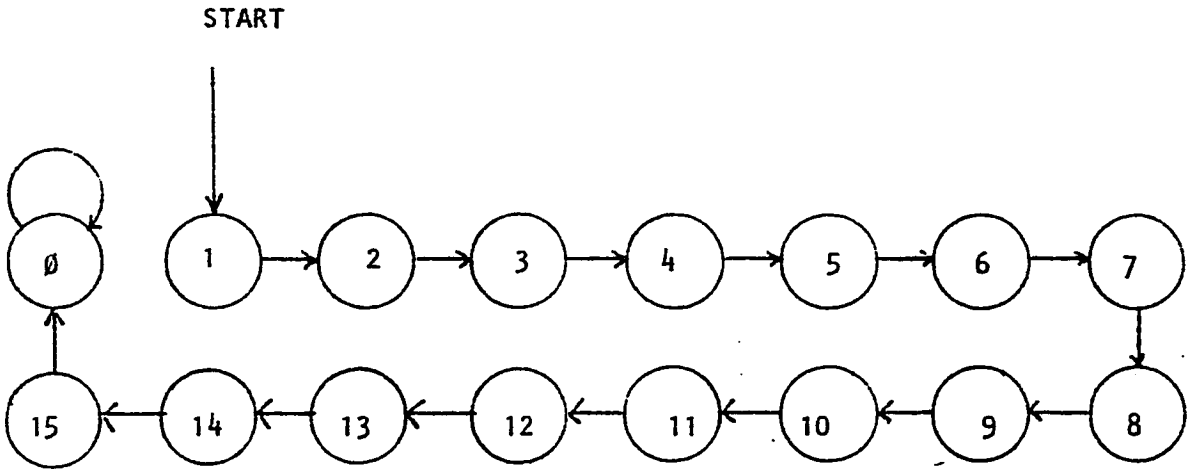


FIGURE 3.23:LOCAL COUNTER STATE-DIAGRAM

The input to the counter is:

1)Start signal from the master controller.

2)Clock-in signal

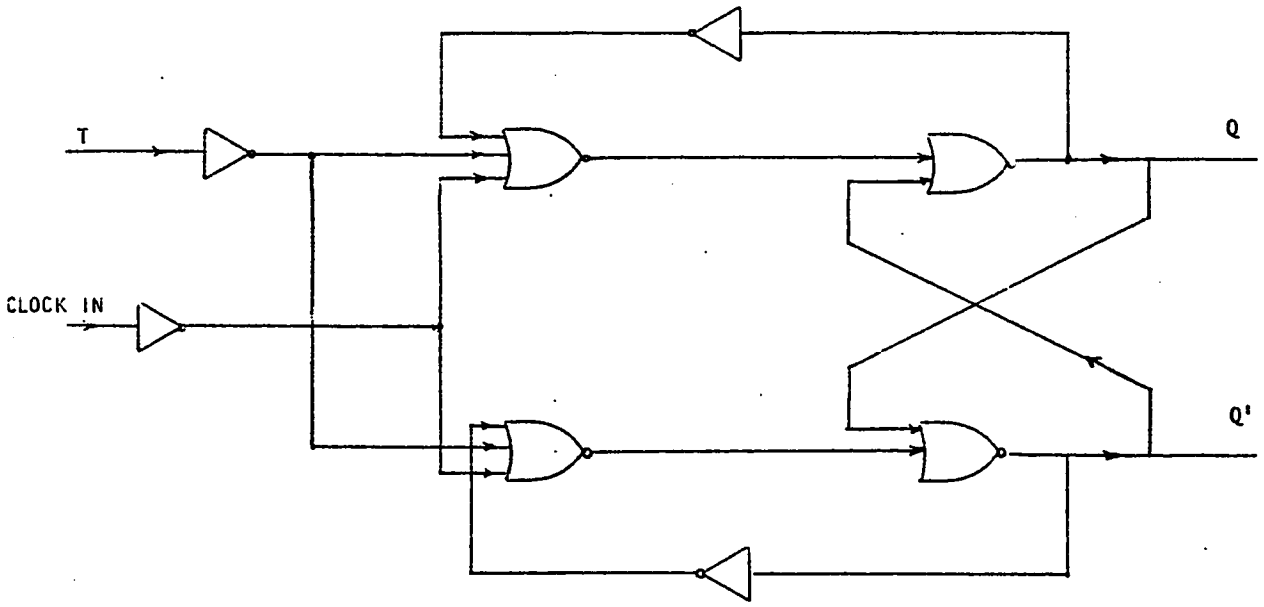
The output of the counter are the count bits Qa, Qb, Qc and Qd. The block diagram of the counter is shown in Figure 3.22. The counter is implemented by using static T flip-flops. Figure 3.24 describes the gate-level description of the T flip-flop.

The clock-in line would be conditionally connected to phi 2 because the fetch unit would conditionally enable the counter for incrementing by one. This is so since the present value of the counter is used in phi 1. Hence, the counter should be updated for the next phase in phi 2. The Ta logic, Tb logic, Tc logic and Td logic is shown in Figure 3.25.

3.5.3 HARDWARE IMPLEMENTATION OF THE FETCH UNIT

The fetch unit implementation is shown in Figures 3.22 to 3.32. The following convention is followed in these designs:

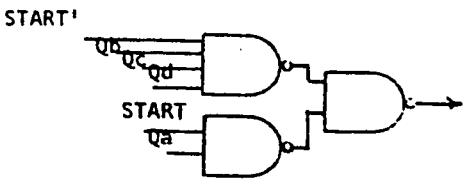
* A single bit line is represented by a thin line with a slash indicating the number of bits.
* More than 1 bit line is represented as a double line with a slash indicating the number of bits.
* A pass-transistor between the parallel bit lines denote that each bit line has one pass transistor with the same control.
* Each figure is represented on a single page. Connections to the other part of the circuit is represented as a block indicating the



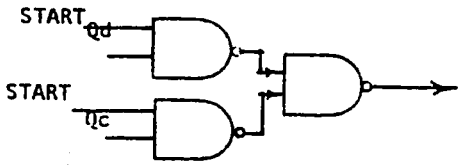
Note:When the clock in is LOW, the T/FF preserves its previous output by feedback. All elements are static. No dynamic structure is used in the design.

FIGURE 3.24:T-FLIP FLOP (Static)

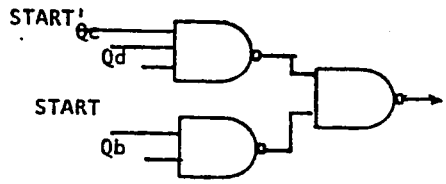
T_a LOGIC:



T_c LOGIC:



T_b LOGIC



T_d LOGIC:

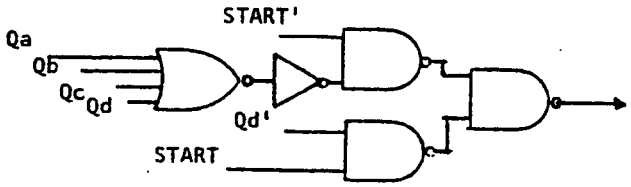


FIGURE 3.25:COUNTER LOGIC

figure number of the circuit.

* Single lines (one or more) taken out from the double lines marked with dots represent special bit lines out of the parallel lines.

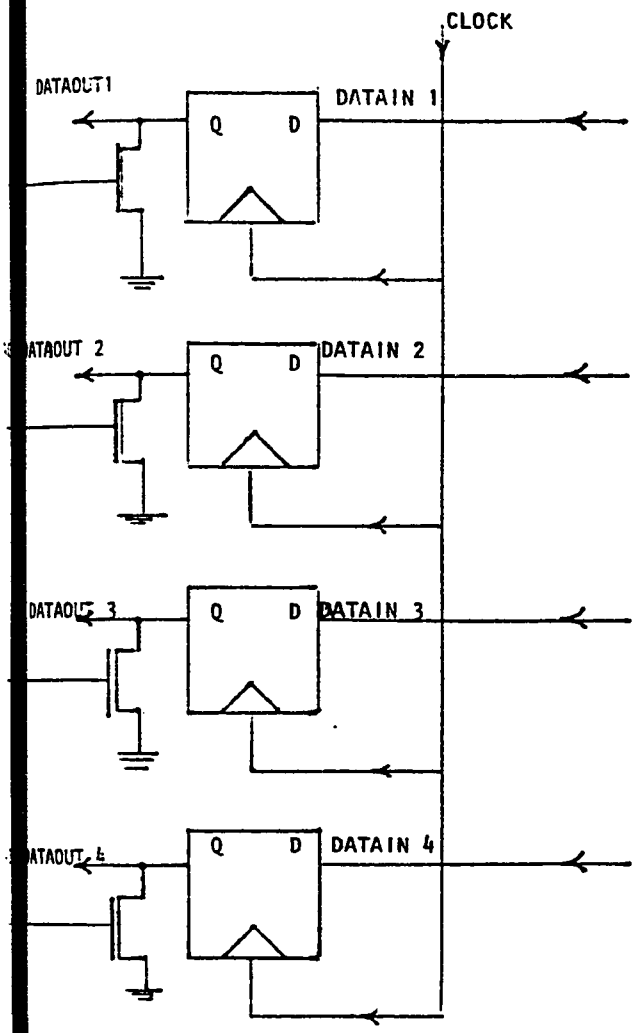
3.5.4 DESCRIPTION OF THE IMPLEMENTATION

Figures 3.22 to 3.25 describe the design of the local counter which resides in the fetch unit. Figure 3.26 describes another local unit; a 4-bit register. Both the counter and the register units are implemented as static units. If these units are to be implemented in dynamic logic, an additional delay will be created in the overall pipeline architecture of the DFEE system.

The new data values are loaded in these units by the fetch unit during phi 2. The fetch unit reads the previous state of these units during phi 1, along with the other inputs it receives from the IC Address Queue and the Valid Address Queue.

Value transfer in these units are done during phi 2 since the present values of these units are used in phi 1 in the fetch unit logic. Since the transfer value (load value) is given to these units at phi 1 and loaded at phi 2, provision is made in the form of inverters and gates which would store the charge till phi 2 is enabled. This is shown in Figure 3.28 and Figure 3.30.

In Figure 3.28 and Figure 3.30, three logic blocks are shown together with there inputs. An instruction is read from the memory cell block either in Figure 3.28 or in Figure 3.27.



START (from M/C)
initially setting the output to LOW.

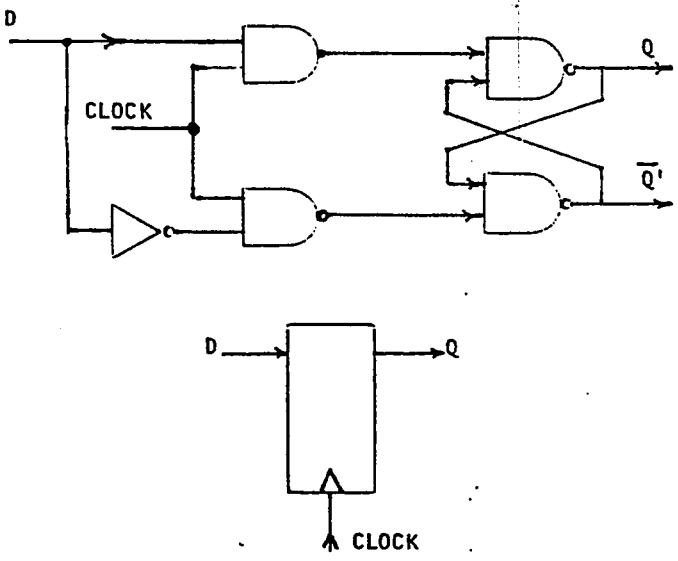


FIGURE 3.26:4-BIT PARALLEL-IN REGISTER (static)

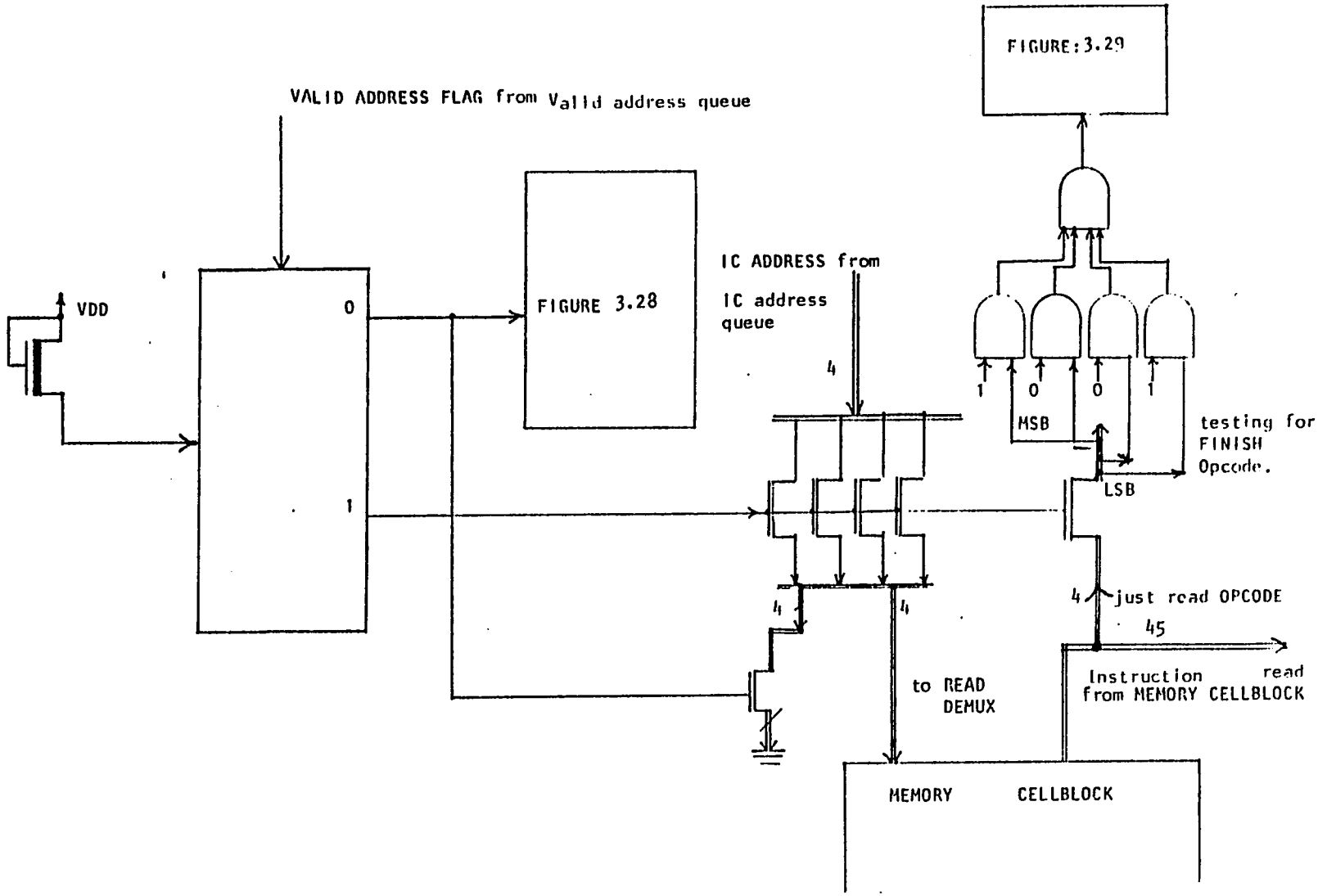


FIGURE 3.27:HARDWARE IMPLEMENTATION OF THE FETCH UNIT

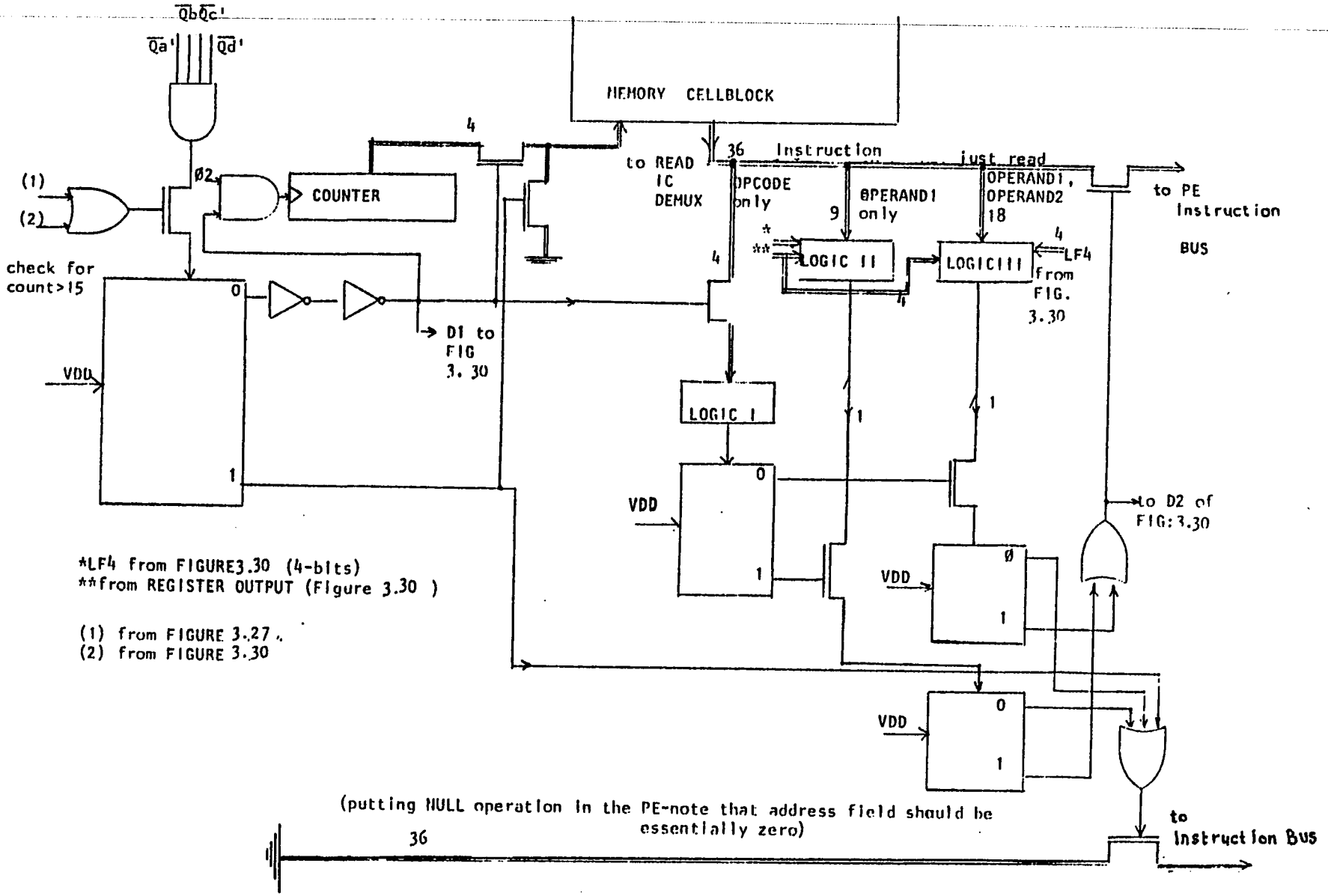


FIGURE 3.28: FETCH UNIT (continued)

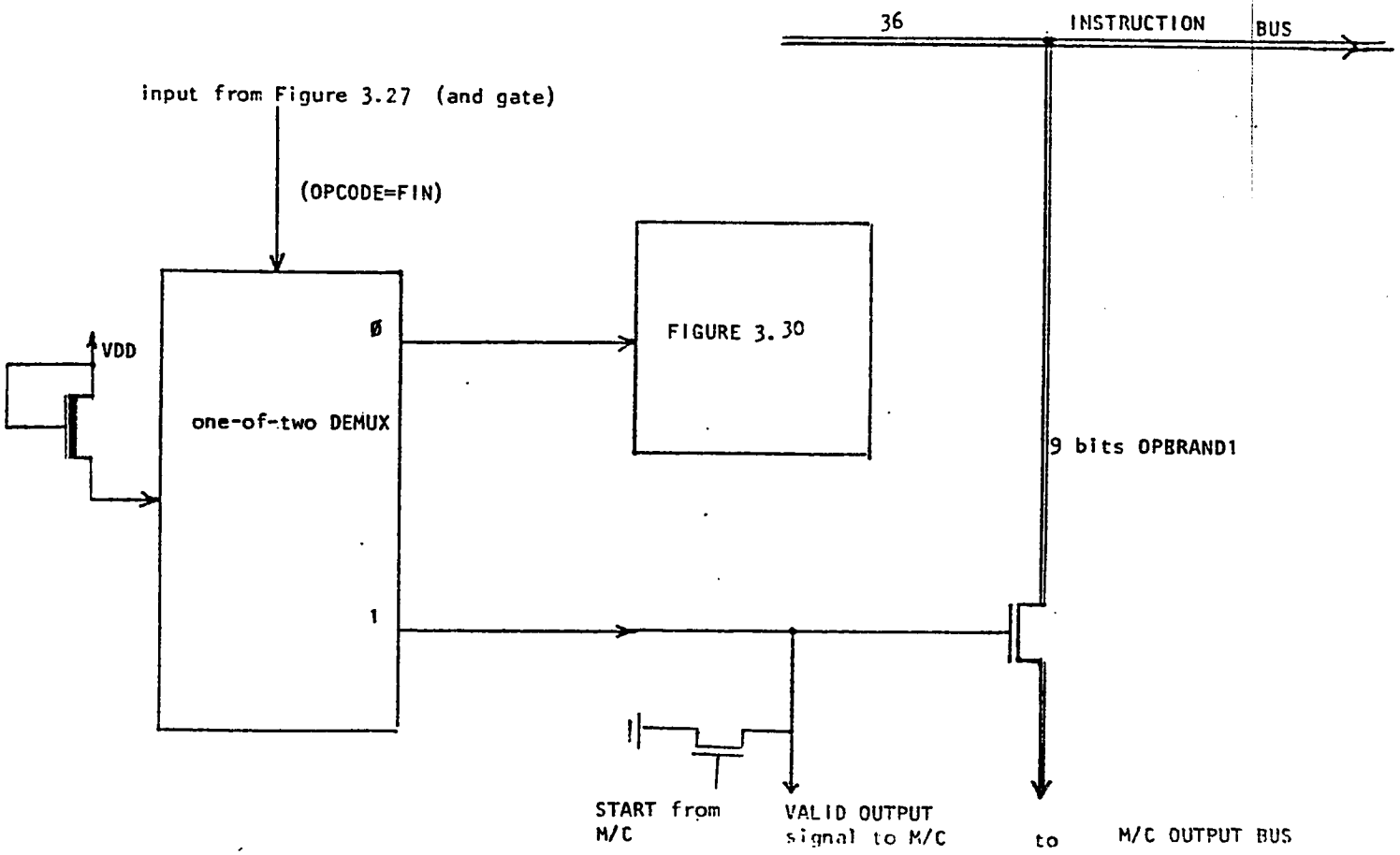


FIGURE 3.29: FETCH UNIT (continued)

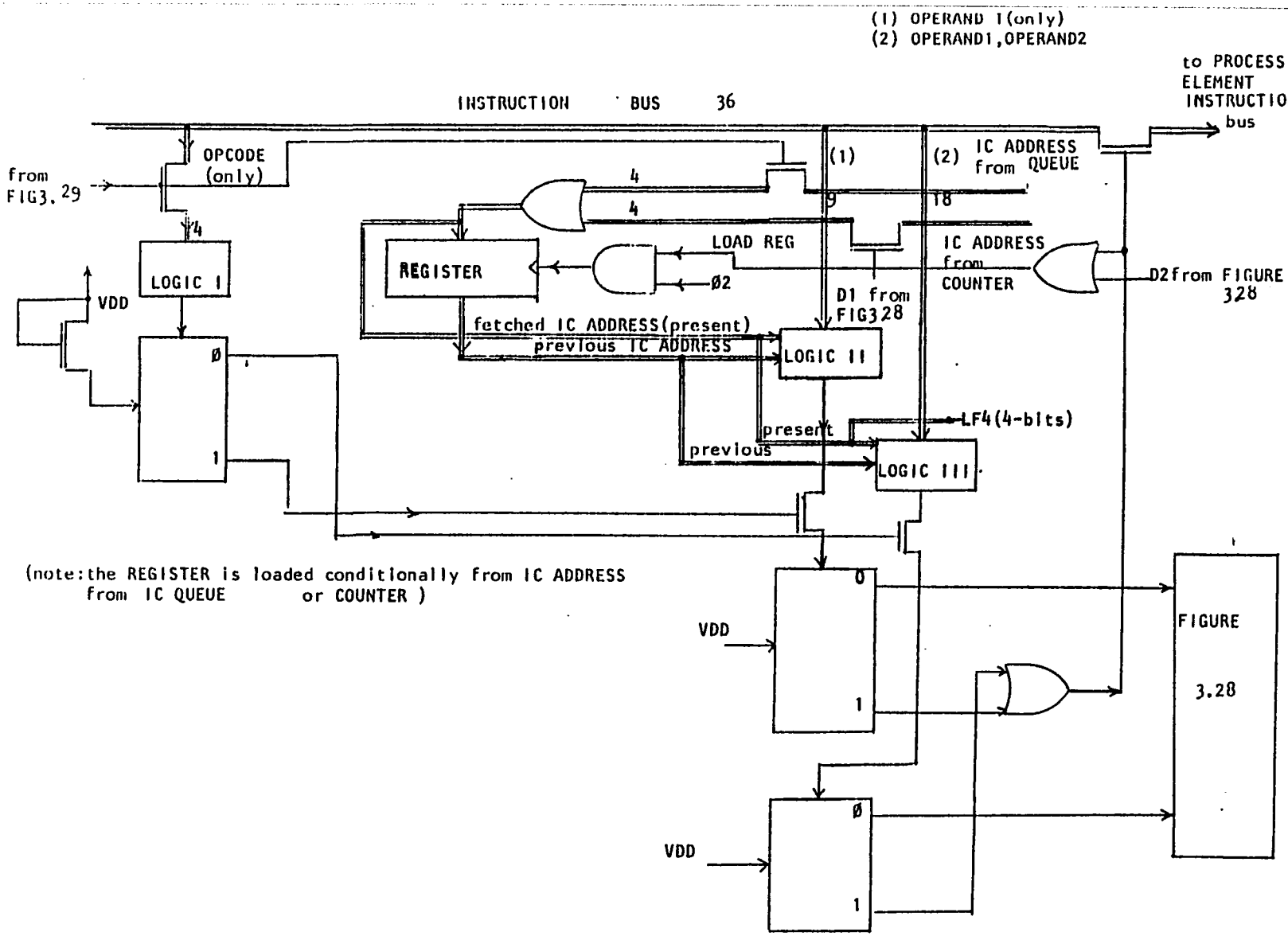


FIGURE 3.30: FETCH UNIT (continued)

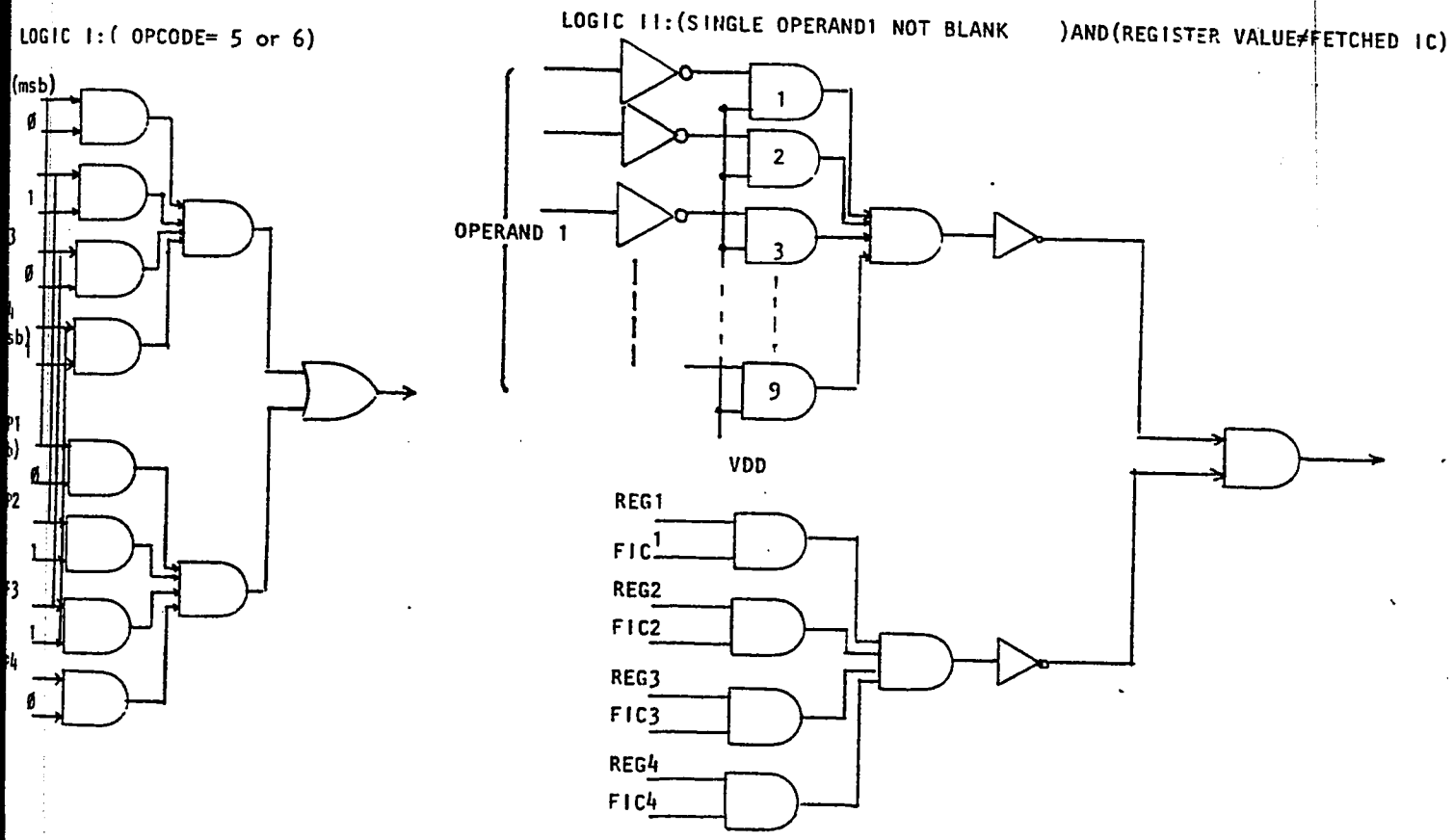


FIGURE 3.31:LOGIC I and LOGIC II-GATE LEVEL DESCRIPTION

LOGIC III:(OPERAND 1 and OPERAND 2 NOT BLANK)AND(REGISTER IS≠ FETCHED IC ADDRESS)

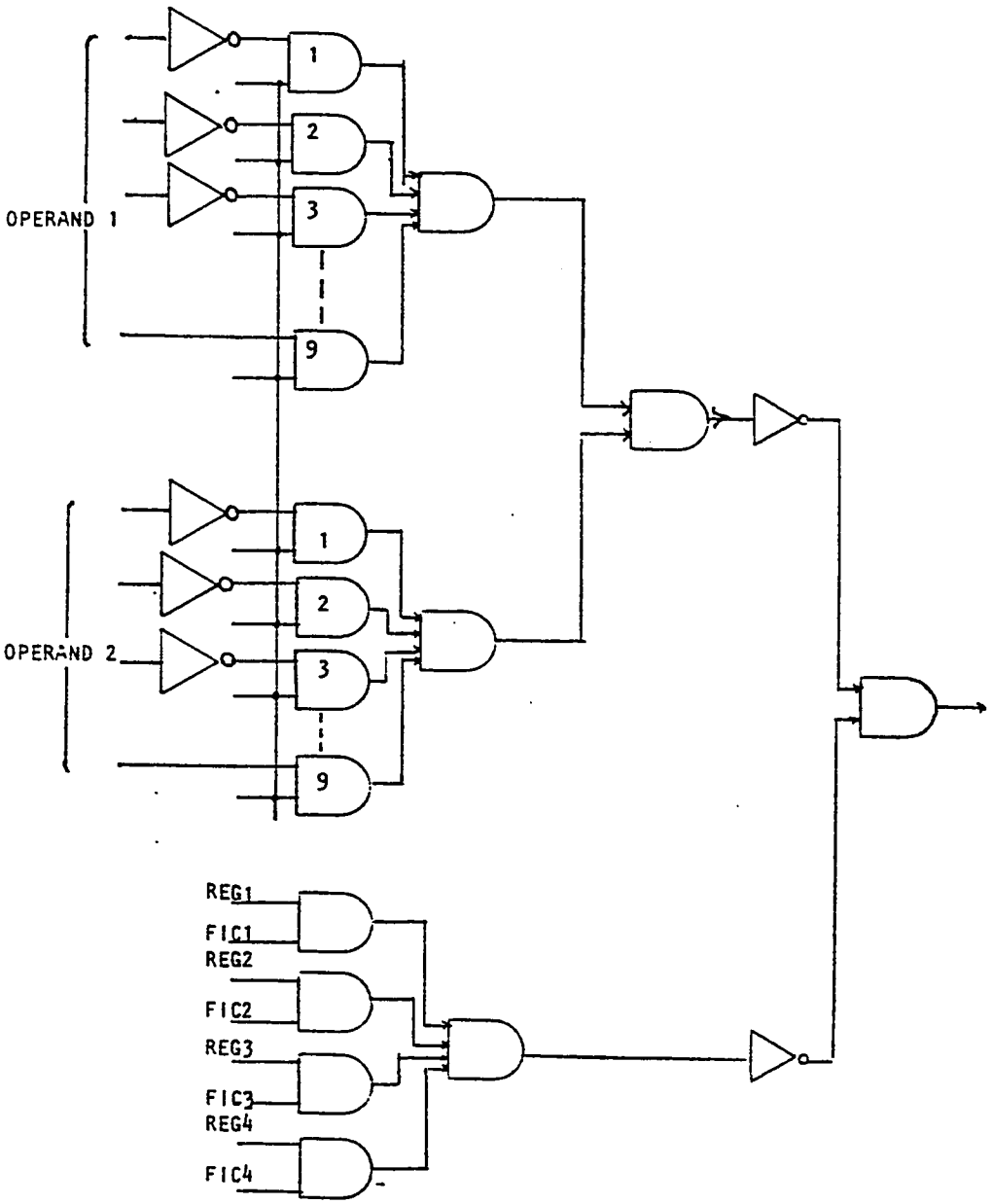


FIGURE 3.32:LOGIC III-GATE LEVEL DESCRIPTION

In Figure 3.28, either the instruction fetched from the cell block is given to the processing element or a blank instruction is given to the processing element.

CHAPTER 4

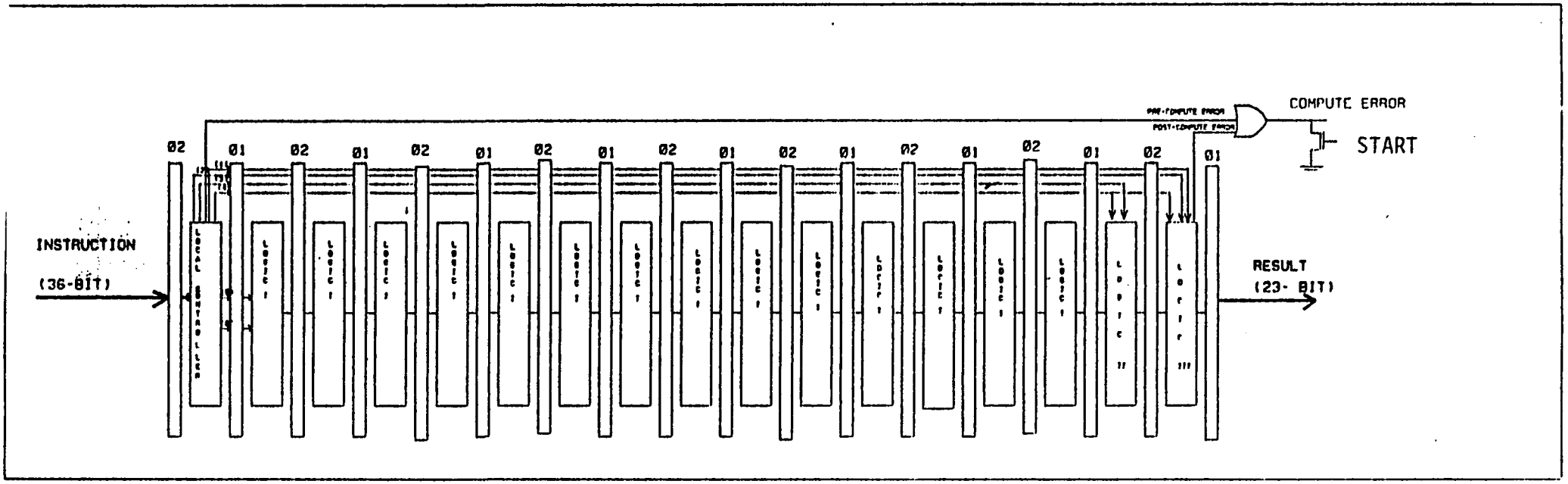
PROCESSING ELEMENT AND MASTER CONTROLLER

The processing element takes an instruction consisting of 36 bits and performs the operation specified by the opcode. The processing element is a pipeline of 18 stages. The computation is done in fixed-point signed-magnitude arithmetic. The output of the processing element is the result field consisting of 23 bits. The result field contains the 9-bit result and the destination address of the result. Figure 4.1 shows the basic input/output of the processing element.

The processing element is described with respect to Figure 4.1.

INPUT: Instruction field.

The opcode, Operand 1 and Operand 2 are used by the processing element for computation. The STAR bits are transmitted unchanged through all the 18 stages of the pipeline. These bits are conditionally changed in Logic III for the case of a LOOP opcode. The 6-bit ADDR1 and ADDR2 are transmitted without being changed by the processing element. These bits go only through the synchronization registers of the pipeline stages without entering the logic blocks.



(1) Loop bit (1-bit)
(2) ADDR1, ADDR2, Star(14-bits).
(3) Sign operand A, sign operand B, M bit.(3-bits)
(4) Opcode (4-bits)
(5) Operand 1, 2 without sign bit (16-bit).
(6) Control parameters.

FIGURE 4.1:PROCESSING ELEMENT PIPELINE

OUTPUT: Result field.

The result field contains the 9-bit result. The result bits go through Logic I, Logic II and Logic III. The ADDR1, ADDR2 bits remain unchanged. The STAR bits may change for the LOOP opcode.

4.1 REGISTERS USED BETWEEN LOGIC BLOCKS

The logic modules are separated by dynamic synchronization registers. Each register consists of two inverters and a pass-transistor. These register banks are alternatively clocked at phi 1 and phi 2. A dynamic two-phase pipeline is used in the design [20].

4.2 SUB-UNITS INSIDE THE PROCESSING ELEMENT

The processing element consists of :

a) Logic I.

b) Logic II.

c) Logic III.

d) Local controller.

Logic I is the heart of the computing process in the DFEE system. It has two input ports and two output ports. It performs ADD, SUBTRACT, MULTIPLY, DIVIDE, SQUARE and SQUARE-ROOT operations in signed-integer arithmetic. Logic I outputs the magnitude part of the result, while the sign part is generated in Logic II. For division and square-root operations, the quotient is taken as the result and the remainder is ignored.

Logic I is fully array-structured to support pipelining. The array design is an extension of the work done by A. Kamal [11]. This array uses controlled adder-subtractor arithmetic cells and control logic cells. The restoring method of division and square rooting is used and the right shift method for multiplication and squaring has been used [11]. The array can do MULTIPLICATION, DIVISION, SQUARE and SQUARE-ROOT operations.

For the case of ADDITION/SUBTRACTION operations, the adder-subtractor arithmetic cells are used for signed magnitude addition/subtraction.

The signed-magnitude addition/subtraction operation can be divided into three parts [19].

a)Pre-complementation of one of the operands depending on the sign bits of operand A, operand B and the operation to be performed.

b)Addition of the operand A and the conditionally modified operand B.

c)Post-complementation of the result depending on the logic values obtained from the full adder circuits.

Let M=0 represent addition operation and M=1 represent the subtraction operation. The equation for the ith sum output S_i from the adder-subtract cell is given by [19]:

$$S_i = A_i \oplus (B_i \oplus XGLB) \oplus C_i$$

where A_i, B_i are the ith bit of operand A and B. C_i is the carry-in from the previous stage and XGLB is the conditional complement used for making the add/subtract cell behave as a subtractor.

For the case of signed-magnitude addition/subtraction operation, the XGLB input conditionally complements the ith bit of operand B. The logic equation for XGLB is given by [19]:

$$XGLB = A_s \oplus B_s \oplus M$$

where A_s is the sign bit of operand A, B_s is the sign bit of operand B.

The pre-complementing and the post-complementing is done outside Logic I since Logic I contains the add/subtract cells.

The local controller inputs the bits which initialize the logic I, i.e., control parameters for the Logic I array. The magnitude part of the operands are inputted into Logic I by the local controller.

The local controller evaluates the pre-compute error from the instruction received.These errors are negative square-root error and divide-by-zero error.

The local controller also inputs into the processing element pipeline the sign of operands, opcode , Star , ADDR1, ADDR2 and two other bits, the LOOP bit and the M bit (Figure 4.1). The LOOP

bit is used for loop opcode in Logic III while the M bit is used in Logic II for add/subtract post-complementation. These bits are in synchronization with the partial results flowing in the logic modules. However, these bits do not go through the Logic I module. These bits are the input to Logic II and Logic III (Figure 4.1). As a result, the fetch unit can input the instruction without waiting for the processing element to complete the previous operation.

Logic II evaluates the sign of the result output from Logic I. Logic III puts the final changes in the result linked with the LOOP and COMP opcodes. Logic III also evaluates possible errors in the result with respect to opcodes and gives an error signal to the master controller. The error given out by the logic III is after the computation and it is called post-compute error. The pre-compute error and the post-compute error are ORed and a single bit output is used as an error indicator to the master controller.

4.3 DESCRIPTION AND IMPLEMENTATION OF LOGIC I

Figure 4.2 describes the basic array structure of logic I. The 15-th stage inputs the result into logic II.

There are four basic cell types in the array structure. Figures 4.3 to 4.6 specify the gate level description of these cells. These gates can be implemented by using a two-phase logic with phi 1 as

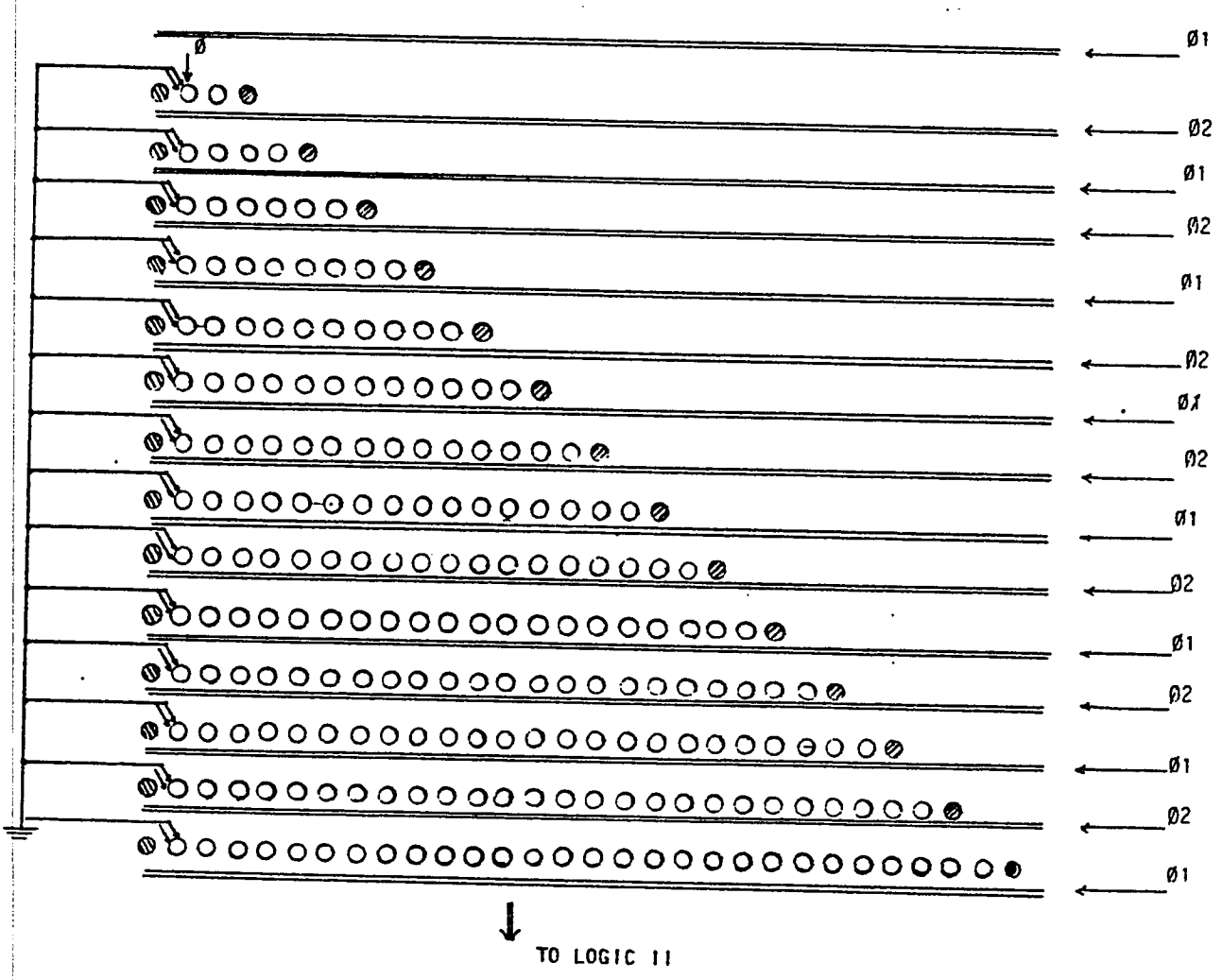
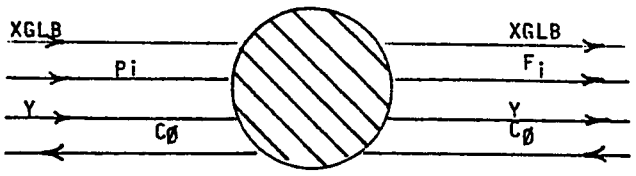


FIGURE 4.2:LOGIC I ARRAY (cell type)



$XGLB := XGLB;$

$F_i := C_{\emptyset}.XGLB + P_i.XGLB + Y$

$C_{\emptyset} := C_{\emptyset};$

$Y := Y;$

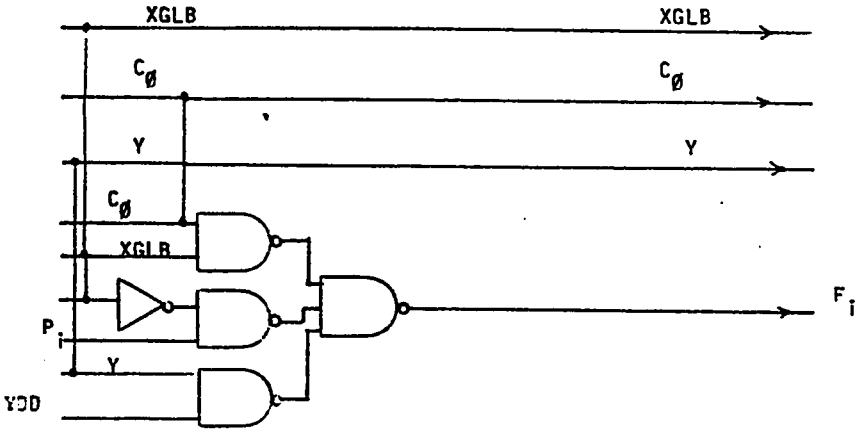
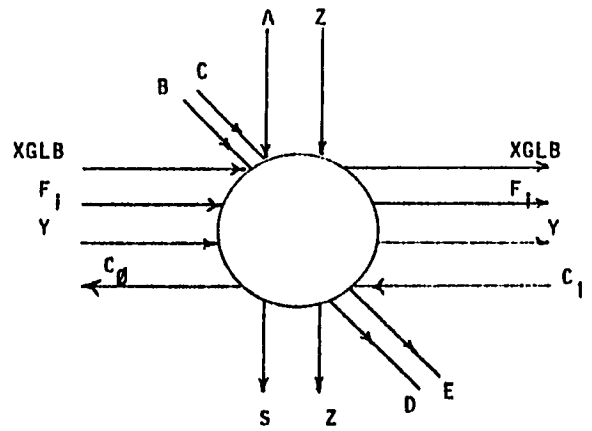


FIGURE 4.3:CELL TYPE-a



$Y := Y;$

$XGLB := XGLB;$

$S := (A \oplus (B \oplus XGLB) \oplus C_1).F_i.\overline{Z} + A.\overline{F}_i.\overline{Z}$

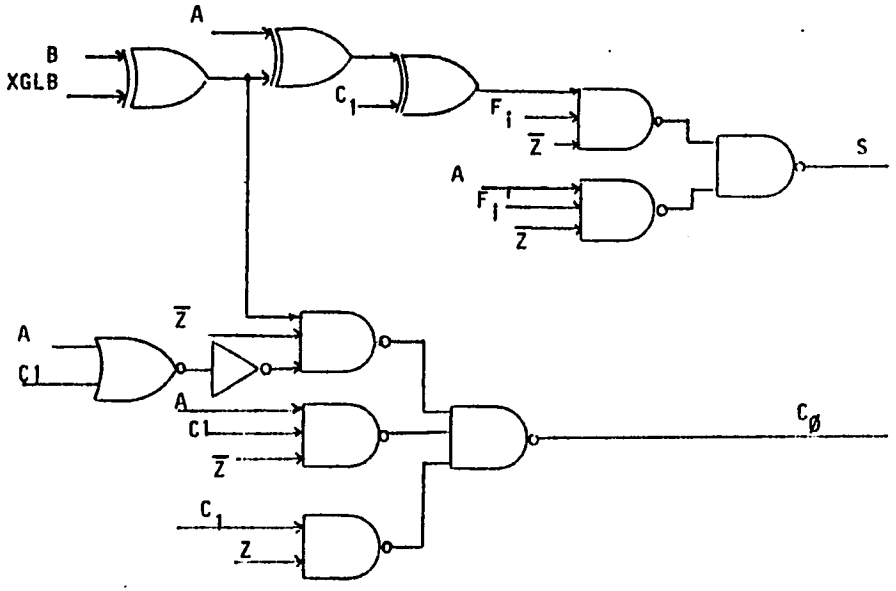
$C_\emptyset := (B \oplus XGLB).(A + C_1).\overline{Z} + A.C_1.\overline{Z} + C_1.Z$

$D := C(B + F_i)\,\overline{Y} + XGLB.Y$

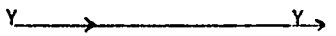
$E := (B + C.F_i).\overline{Y}$

$F_i := F_i$;

S:



$C_\emptyset$:



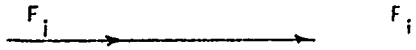


FIGURE 4.4:CELL TYPE-b

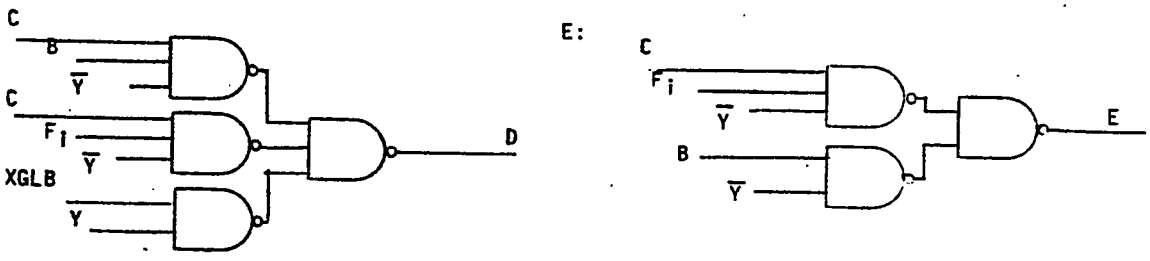


FIGURE 4.4:CELL TYPE-b (continued)

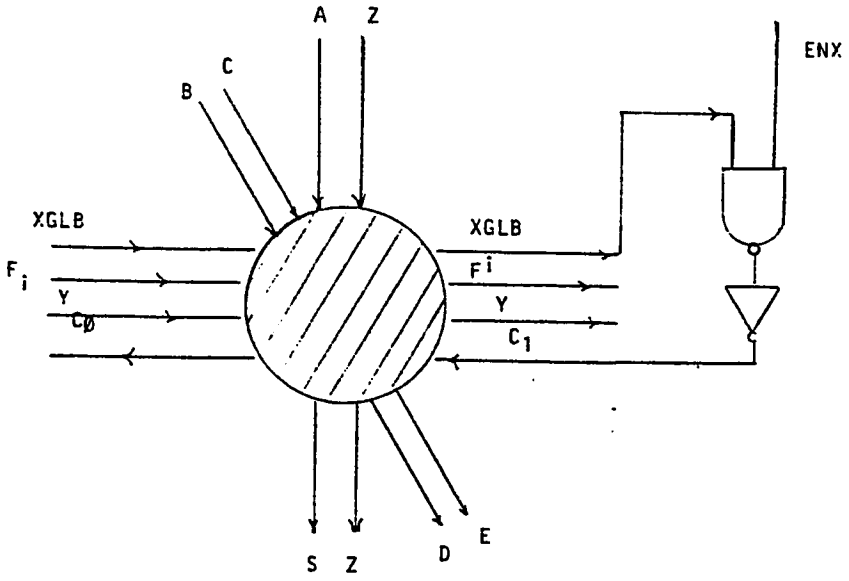
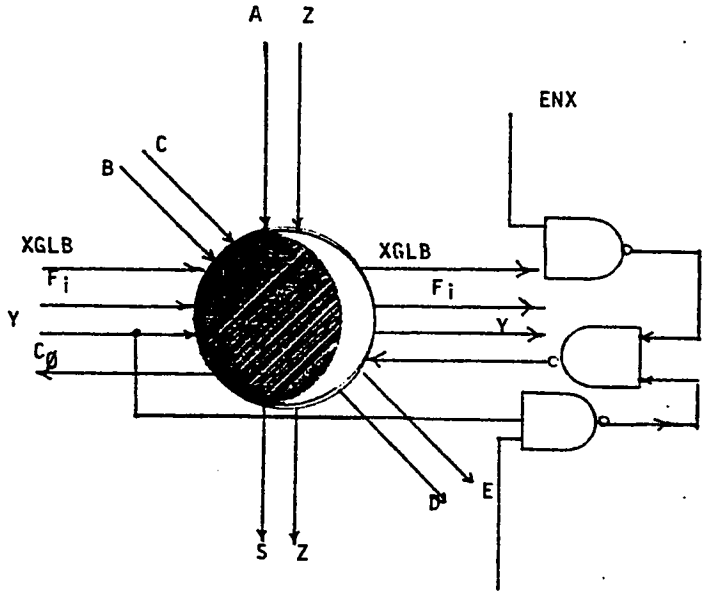


FIGURE 4.5 :CELLTYPE (output defined in Fig 4.4 b) c)



carryout bit from last stage

FIGURE 4.6 d)CELLTYPE

the pre-charge and evaluation phase, and with phi 2 as the transfer phase to another stage (see Figure 4.7)[18].

Figure 4.8 shows the input/output configuration of Logic I. Figure 4.8 also illustrates the synchronization of the two input ports and the two output ports. Each square block represents the dynamic synchronization register with the phase written inside it. The logic modules which lie between the synchronization registers in Logic I are not shown in this figure.

Synchronization between the data coming from the Input-port 1 and Input-port 2 is achieved by delaying the register banks such that the data merge at the destination logic cell in synchronization. Similarly, to obtain the output after 15 clock phases at both output ports, additional delay register banks have been placed at the Output-port 2.

Figure 4.8 shows that Port 2 must be used as an input port as well as an output port. This requirement comes from Logic I where the same logic cell requires an input for one opcode after it gives output for another opcode. Therefore, Port 2 receives and outputs data alternatively at phi 1 and phi 2.

Figure 4.9 gives the input format for Logic I. The inputs include two sets:

a) Operands: These operands are input through Input Port 1 or 2 according to the opcode. None of these operand inputs should have an undefined value. Either logic-0 or logic-1 should be connected to these inputs.

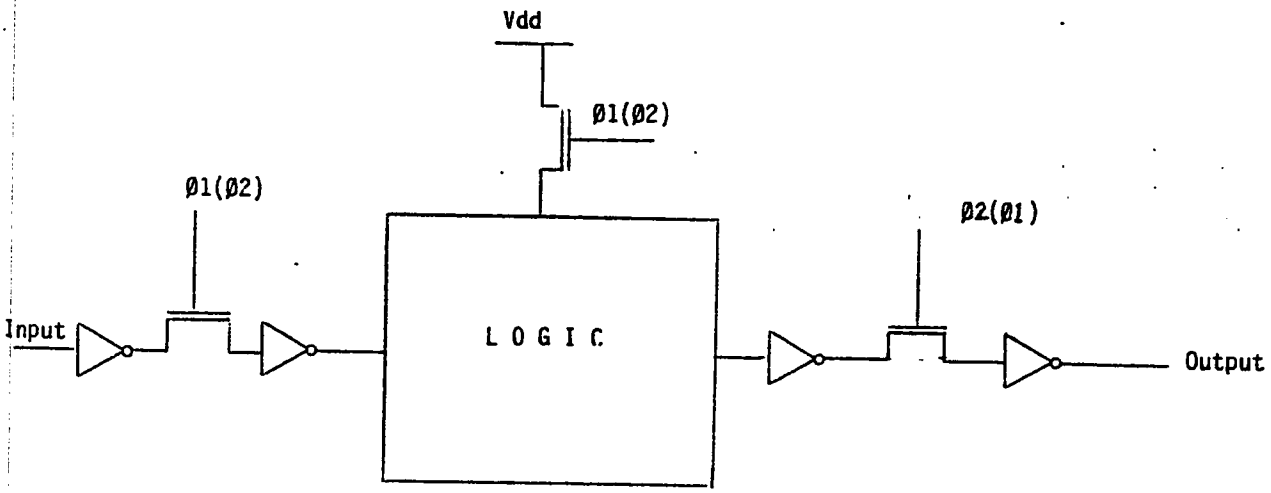


FIGURE 4.7

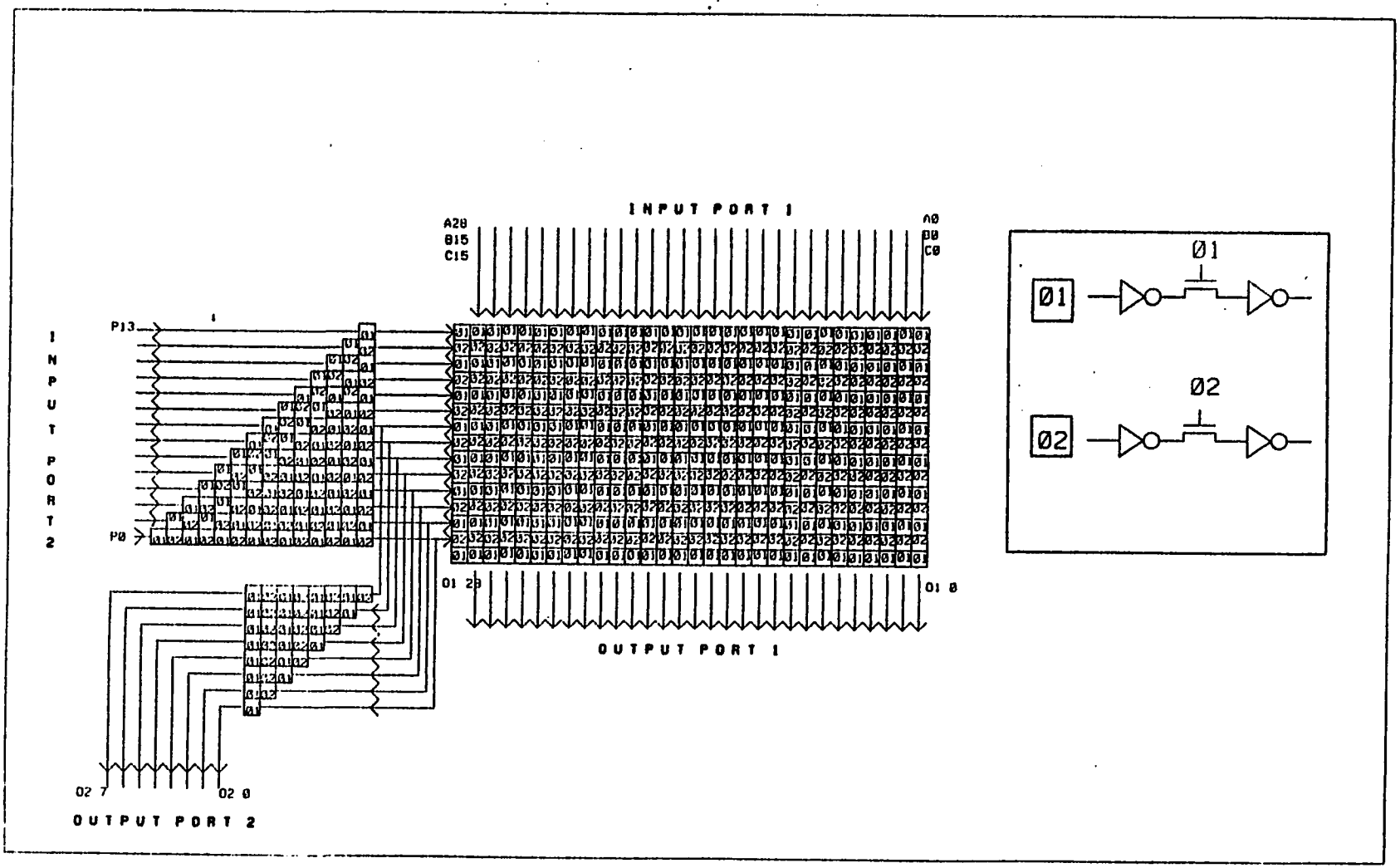


FIGURE 4.8: INPUT/OUTPUT CONFIGURATION OF LOGIC I ARRAY (with synchronization registers)

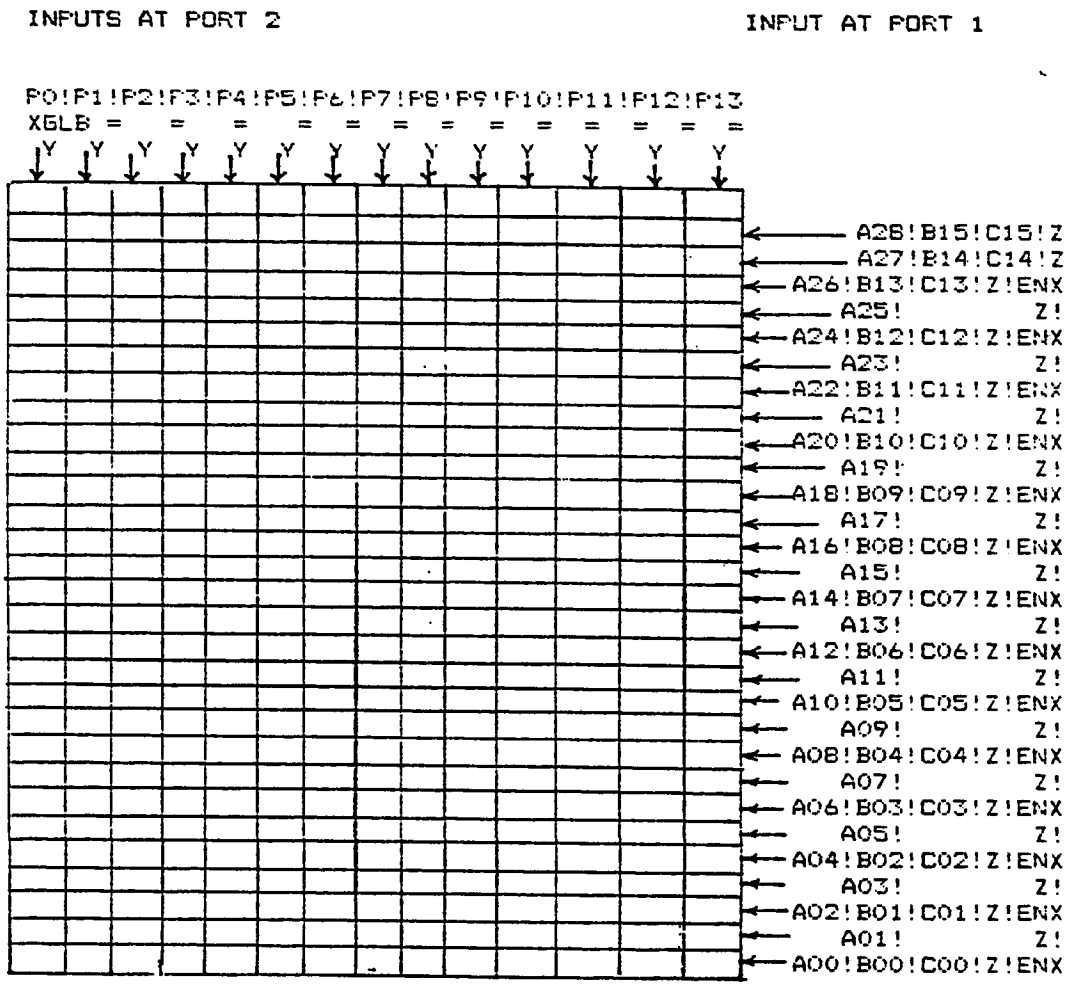


FIGURE 4.9: INPUT FORMAT

The operands are inputted in these vectors (Figure 4.9):

A0,A1,...,A28

B0,B1,...,B15

C0,C1,...,C15

P0,P1,...,P13

b) Control Parameters: These parameters initialize the Logic I for the specific opcode. They make the logic I array cells multi-functional. The logic of the cell is used for different operations by these control parameters. The description of the control parameters follows (Figure 4.9):

FOR ADD/SUBTRACT OPERATIONS:

Y=1

For the Add/Subtract operation, Z vector has the following configuration:

Z(2i):=0; 0<=i<=8

Z(2i+1):=1; 0<=i<=7

Z17=,...,Z28=1;

ENX=0

XGLB= As @ Bs @ M

FOR OTHER OPERATIONS

Y=0;

Z00=0,Z01=0,...,Z28=0;

XGLB=1 for square-root/divide operations.

XGLB=0 for multiply/square operations.

4.4 INPUT/OUTPUT FORMAT OF LOGIC I

The following description gives the input ports, the input bit pattern of operands, the control parameters and the output ports for each respective opcode. The result out of this Logic I array is unsigned. The sign part is determined in Logic II.

CONVENTION:

Output-port 1 has bit 0 to 28. The bits in Output-port 1 will be represented as:

O1 00:Output port 1, bit 00.

O1 01:Output port 1, bit 01

and so on.

Output-port 2 has bit 0 to 7. These bits will be represented as O2 00:Output-port 2, bit 00 and so on.

ADD/SUBTRACT

M bit will be used by the local controller to distinguish between add (M=0) and subtract (M=1).

Control Parameters:

Y=1

Z(2i)=0; for 0<=i<=8;

Z(2i+1)=1; for 0<=i<=8;

Z(i)=1; for 18<=i<=28

ENX=0;

XGLB= As @ Bs @ M

Operand format:*A +/- B*

Operand A:

A(2i)=ith bit of operand A, 0<=i<=7, A(00) is the LSB.

A(2i+1)=0 for 0<=i<=6

A15=...=A28=0;

Operand B:

B(i)=ith bit of operand B, 0<=i<=7, B(0) is LSB

B08=...=B15=XGLB(The logic value determined by the local controller).

C0=...=C15=0; P0=...=P13=0;

INPUT PORTS USED:

Both Input-port 1 and 2.

OUTPUT-PORT: 1

OUTPUT BITS: O1(2i) for 0<=i<=7, O1 (00) is the LSB.

The carry-out is generated from O1 16 to be used by logic II.

MULTIPLY

Operand 1 x Operand 2

Control Parameters:

Y=0;

Z=0;

ENX=1;

XGLB=0;

Operand 1:

P(i)=ith bit of operand 1, For 0<=i<=7, P(0) is the LSB.

P8=...=P13=0;

Operand 2:

B0=...=B7=0;

B(8+i)=ith bit of operand 2. For 0<=i<=7. B(8) is the LSB.

C0=B0, C01=B01,...,C15=B15

A0=...=A28=0;

INPUT PORT: Port 1 and 2

OUTPUT PORT:1 O1(i), 8<=i<=15 , O1 08 is the LSB.

DIVIDE

Operand 1/ Operand 2

Only the quotient is output and the remainder is ignored.

Control Parameters:

Y=0;

Z=0;

ENX=1;

Operand 1:

A0=...=A7=0;

A(8+i) =ith bit of operand 1,For 0<=i<=7, A(8) is LSB.

A9=...=A28=0;

Operand 2:

B0=...=B7=0;

B(8+i)=ith bit of operand 2, For 0<=i<=7, B(8) is the LSB.

C0=B0, C01=B01,...,C15=B15

P0=...=P13=0

INPUT PORTS:1 and 2

OUTPUT PORT:2

O2(j), 0<=j<=7, O2 00 is the LSB.

SQUARE

Control Parameters:

Y=0;

Z=0;

ENX=1;

XGLB=0;

Operand 1:

P(i)=ith bit of operand 1, For 0<=i<=7, P(0) is the LSB.

A00=...A28=0;

B00=...=B13=1;

B14=B15=0;

C14=1,C15=C13=...C00=0;

INPUT PORT: 1 and 2

OUTPUT PORT:1

O1(j) , For 0<=j<=7, O1 (00) is the LSB.

SQUARE ROOT

Control Parameters

Y=0;

Z=0;

ENX=1;

XGLB=1;

Operand 1

A(i)=ith bit of operand 1. 0<=i<=7, A(0) is the LSB.

A08=...=A28=00;

P00=...=P13=0;

B00=...=B13=1;

B14=B15=0;

C14=1,C15=C13=...C00=0;

INPUT PORT:Input-Port 1 and 2

OUTPUT PORT: 2

O2(i), 0<=i<=7, O2(00) is the LSB.

Figure 4.10 shows the input operands and the outputs for all the opcodes. Each input is matched by the output. Figure 4.10 has been verified by extensive logic level simulation. The simulation program is listed in Appendix C.

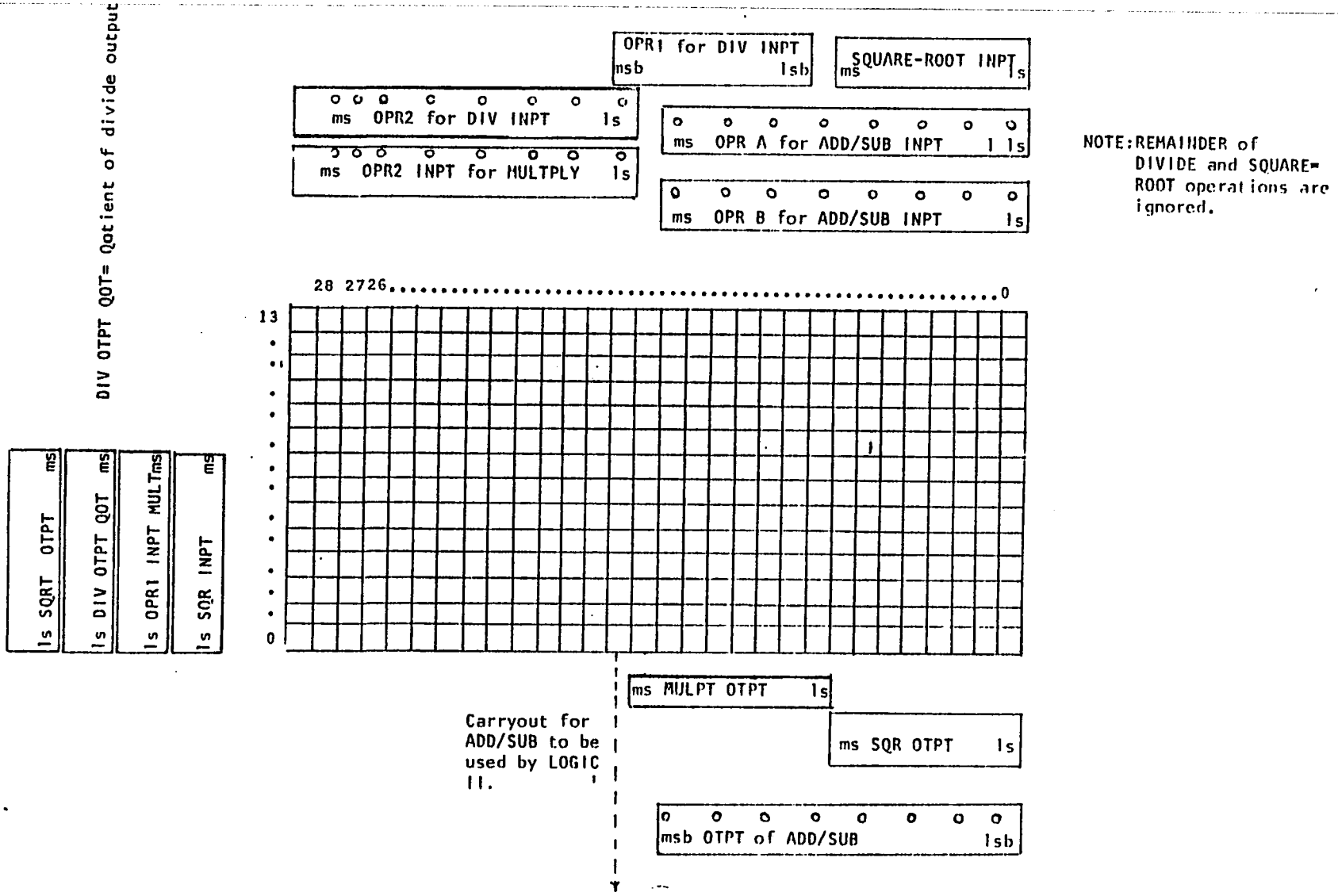


FIGURE 4.10: INPUT/OUTPUT FORMAT

4.5 LOGIC II:DESIGN

The function of Logic II is to correct the result of ADD/SUBTRACT operations, i.e., the post-complement of the result. The logic II takes into account the sign bits of Operand A, Operand B and the carry-out generated by Logic I.

Logic II also generates the overflow indication in the case of ADD/SUBTRACT operations. Further, it generates the sign of the result for all operations.

Figure 4.11 shows the input/output of Logic II. Logic II gets the input from :

a)Local controller.

b)Logic I.

The outputs from Logic II consist of :

1)Result (8-bits)

2)Result sign (1-bit)

3)Overflow indicator.

Figure 4.12 shows the implementation of Logic II. The four-bit opcode coming from the local controller is used as data-selector input. Both the add and subtract operation results go through the logic for add/subtract as shown in Figure 4.12. Only one operation will be true; therefore both the inputs, either for add or subtract, are ORed. For the other opcodes, the result bits from Logic I are transmitted without being changed by Logic II. The opcode is coded as a 4-bit vector:

NOP 0000B 00H

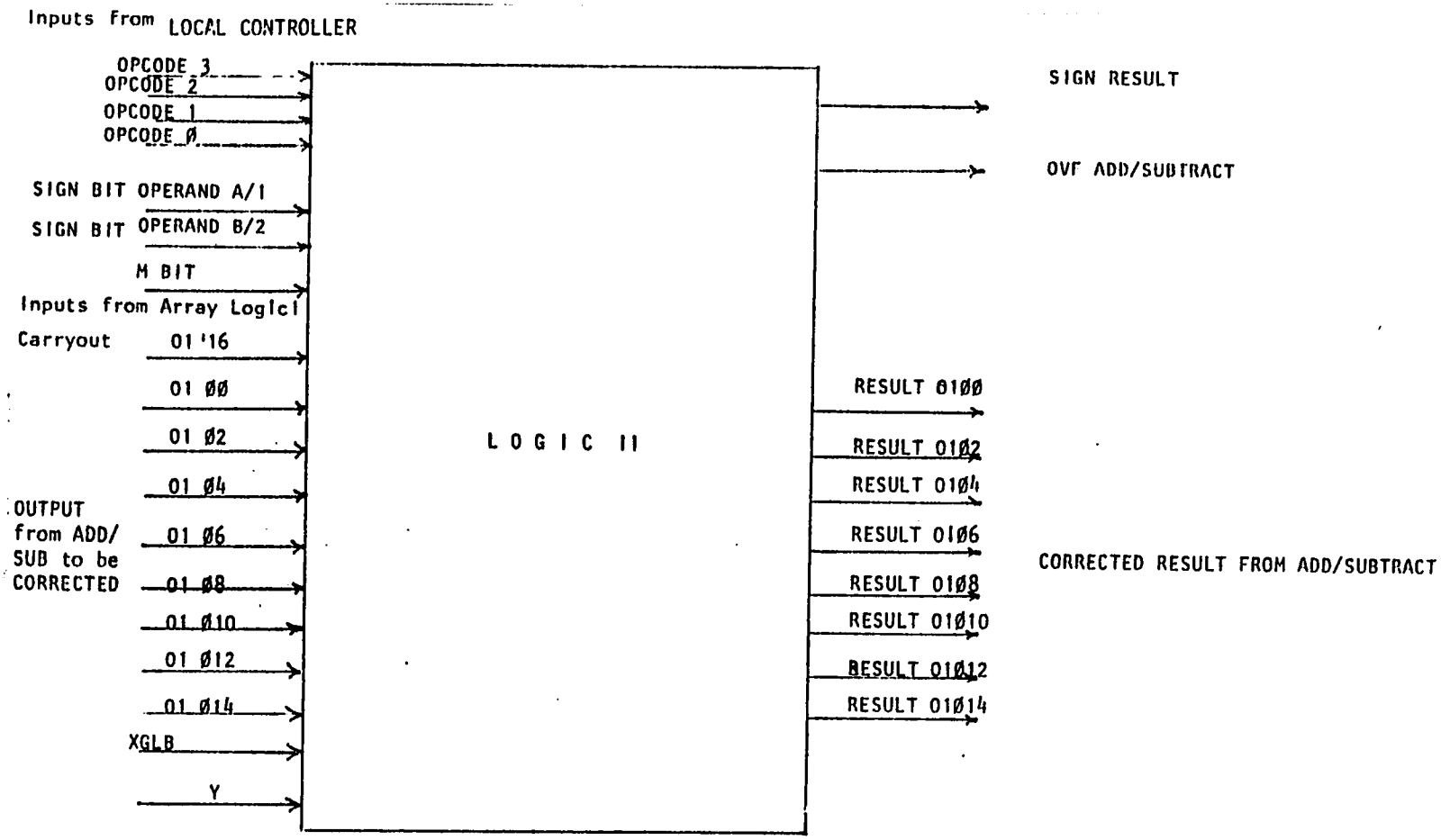


FIGURE 4.11:LOGIC II-INPUT/OUTPUT

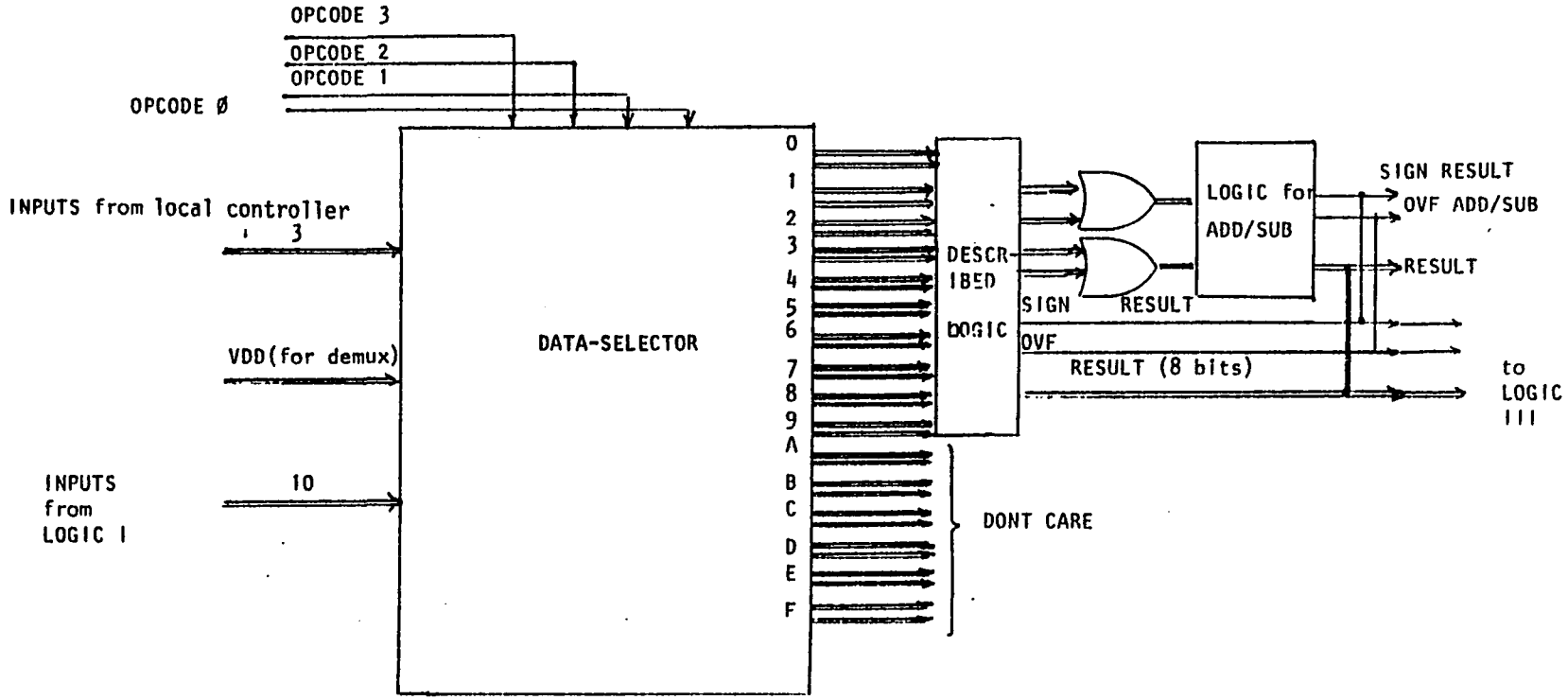


FIGURE 4.12:LOGIC II-HARDWARE BLOCK DIAGRAM

ADD	0001B 01H
SUBTRACT	0010B 02H
MULTIPLY	0011B 03H
DIVIDE	0100B 04H
SQRT	0101B 05H
SQR	0110B 06H
COMP	0111B 07H
LOOP	1000B 08H
FIN	1001B 09H

The FIN opcode will not go through the processing element. It will be detected by the fetch unit for the purpose of terminating the computation.

4.6 DESCRIPTION OF LOGIC II

Note: (* comment statement *)

CASE

```
If opcode =00H then (*nop opcode*)
  Begin
  sign result:=0;
  result field=0;(*all the bits*)
  OVF=0;
  End;
If opcode =01H then (*Add opcode*)
  Enable inputs to LOGIC FOR ADD/SUBTRACT
If opcode=02H then (*Subtract opcode*)
  Enable inputs to LOGIC FOR ADD/SUBTRACT
```

```
If opcode =03H then (*Multiply opcode*)
  Begin
  If (Sign Operand 1) @ (Sign Operand 2) true then
    Sign-result:=1;(*negative*)
  Else
    Sign Result:=0;(*positive*)
  Result Field Out:=Result Field In;(*transfer without modifying*)
  OVF:=0;
  End;
If opcode =04H then (*Divide opcode*)
  Begin
  If  (Sign Operand 1) @ (Sign Operand 2) true then
    Sign Result:=1;(*negative*)
  Else
    Sign Result:=0;(*positive*)
  Result Field Out:=Result Field In;
  OVF:=0;
  End;
If opcode =05H then (*Square root*)
  Begin
  Sign Result:=0;
  Result Field In:=Result Field Out;
  OVF:=0;
  End;
If opcode =06H then (*Square*)
  Begin
  Sign Result:=0;(*positive*)
```

```
   Result Field Out:=Result Field In;
   Ovf:=0;
   End;
If opcode =07H then (*Comp*)
   Begin
   Sign Result:=0;(*always positive number*)
   Result Field In:=Result Field Out;
   OVF:=0;
   End;
If opcode =08H then (*Loop*)
   Sign Result:=Sign Operand 2;(*By defination of loop opcode*)
   Result Field In:=Result Field Out;
   OVF:=0;
   End;
If opcode =09H then
   don't care
```

END CASE

Figure 4.13 describes the LOGIC FOR ADD/SUBTRACT. Logic II corrects the result of add/subtract operation only.

Only one opcode will be true. Therefore, the output (8-bit result), sign result (1-bit) and the OVF line are connected with the other lines as shown in Figure 4.12. Only one out of these lines will have a valid output while the rest of the lines will be in three-state. These lines will not be enabled by the data-selector.

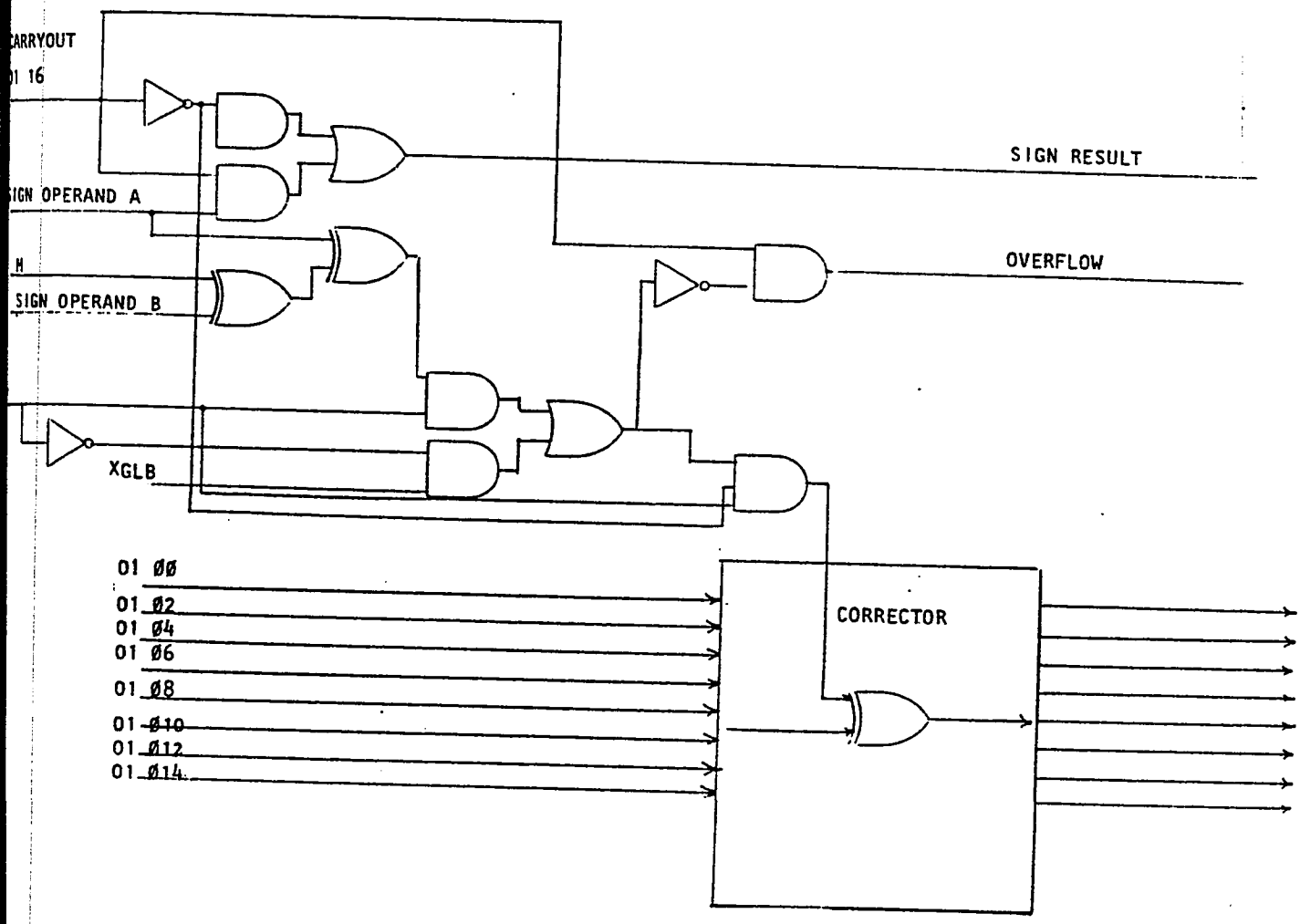


FIGURE 4.13:LOGIC FOR ADD/SUBTRACT

The result bits for the rest of the opcodes are transmitted as they are. The other outputs from Logic I are transmitted as well without modification. These outputs will be used in Logic III where the result will be generated. The other outputs are given as follows:

From output port 2:

O2 00...07

From output port 1:

O1 01,03,05,07,09,11,13,15,16.

4.7 VERIFICATION AND LOGIC LEVEL SIMULATION OF LOGIC I AND LOGIC II

Logic I has been simulated with the gate/logic level description. Each stage of the array with the four types of cells as specified in Figure 4.3 to 4.6 have been simulated. The simulation results were used to verify the input/output configuration given in Figure 4.10.

Each cell was represented as an element in an array structure with boolean inputs and outputs as described by the equations given in the respective figures. This way, the boolean equations were also verified.

Logic II was added into the simulation and add/subtract operations were run on the simulation program. Therefore, the modification to Logic I array structure including the add/subtract operation with overflow detection and corrector were verified.

The simulation program was written in Waterloo PASCAL. Appendix C contains the listing of the program run for the case of add/subtract operations. Appendix C also contains results of multiplication, division and square operations run on the simulation program.

4.8 DESCRIPTION OF LOGIC III

Logic III performs three functions:

a) It generates the result from the outputs received from Logic II and the inputs from the local controller.

b) It updates the result before transfering it to the distribution network for the case of LOOP and COMP opcodes.

FOR COMP OPCODE:

The local controller receives the opcode COMP with the two operands. The compare operation is performed using subtract operation. Operand 1 is subtracted from Operand 2. If the result from Logic II is an all zero vector, Logic III sets the result to an all zero vector, i.e., 000000000B. If the result from Logic II is not an all zero vector, Logic III sets the result to be 000000001B.

FOR LOOP OPCODE:

The loop instruction precedes the compare instruction in execution. The loop instruction contains either 000000000B or 000000001B from the compare instruction. Operand 2 is the result to be routed to either Address 1 or Address 2 depending on Operand 1. The local

controller inputs the opcode to inform Logic III for LOOP and COMP opcodes. The local controller also inputs the loop bit. The loop bit is zero when operand 1 is an all zero vector and the loop bit is one when the Operand 1 is not an all zero vector. The result which comes out of Logic II is Operand 2. It is the result to be conditionally routed to Address 1 or Address 2. Depending on the loop bit, the Star value (2-bits) is set by Logic III to destine the result.

c)Post-compute error evaluation.

Logic III checks for the post compute errors after the result has been output from Logic II. The possible post-compute errors are listed below:

i)ADD/SUBTRACT:In this case, the overflow is detected and the master controller is notified. The overflow bit is received by Logic III from Logic II.

ii)MULTIPLY: For this opcode the overflow is detected in the Output-port 1 by inspecting the next to most significant bit of multiplication output, i.e., O1 16-output port 1 , bit 16.

iii)SQUARE: The overflow is detected by inspecting the next to most significant bit of the square output, i.e., O1 08-output port 1, bit 8.

iv)SQUARE-ROOT AND DIVISION:These errors are detected by the local controller as pre-compute errors.

Figure 4.14 gives the input/output description of Logic III. Logic III has two input sources:

INPUT:

a)Logic II

1)O1 00...16, Output Port 1, bits 0 to 16

2)O2 00...07, Output Port 2, bits 0 to 7

3)Result Sign (1-bit)

4)OVF indicator (1-bit)

b)Local controller

1)Opcode (4-bit)

2)Address 1, 2 (12-bit)

3)Star (2-bit)

4)Loop bit (1-bit)

OUTPUT

1)Result field

a)Result (9-bit)

b)Address 1 (6-bit)

c)Address 2 (6-bit)

d)Star (2-bit)

Total result field =23 bits.

2)Post-compute error indicator:

This is a single bit error indicator which is ORed with the pre-compute error indicator given by the local controller.

Figure 4.15 shows the implementation of Logic III. The opcode bits are used to select the operation. The operation is enabled by the data-selector. Of the sixteen possibilities only 10 are used; the rest are don't care and therefore are set to zero. Since only one of the opcode is true, only one of the error indicator and result field is valid. The other outputs are three-stated. Therefore, all the output lines emerging from logic III are connected to a common bus to give one error indicator and one result field.

4.9 DESCRIPTION OF ERROR INDICATOR AND RESULT PACKET GENERATION IN LOGIC III.

```
CASE
If opcode=00h then (*null operation*)
  Begin
  Result:=000000000b;
  Address 1:=000000b;
  Address 2:=000000b;
  Star:=00b;
  End;
If opcode =01h then (*add operation*)
  Begin
```

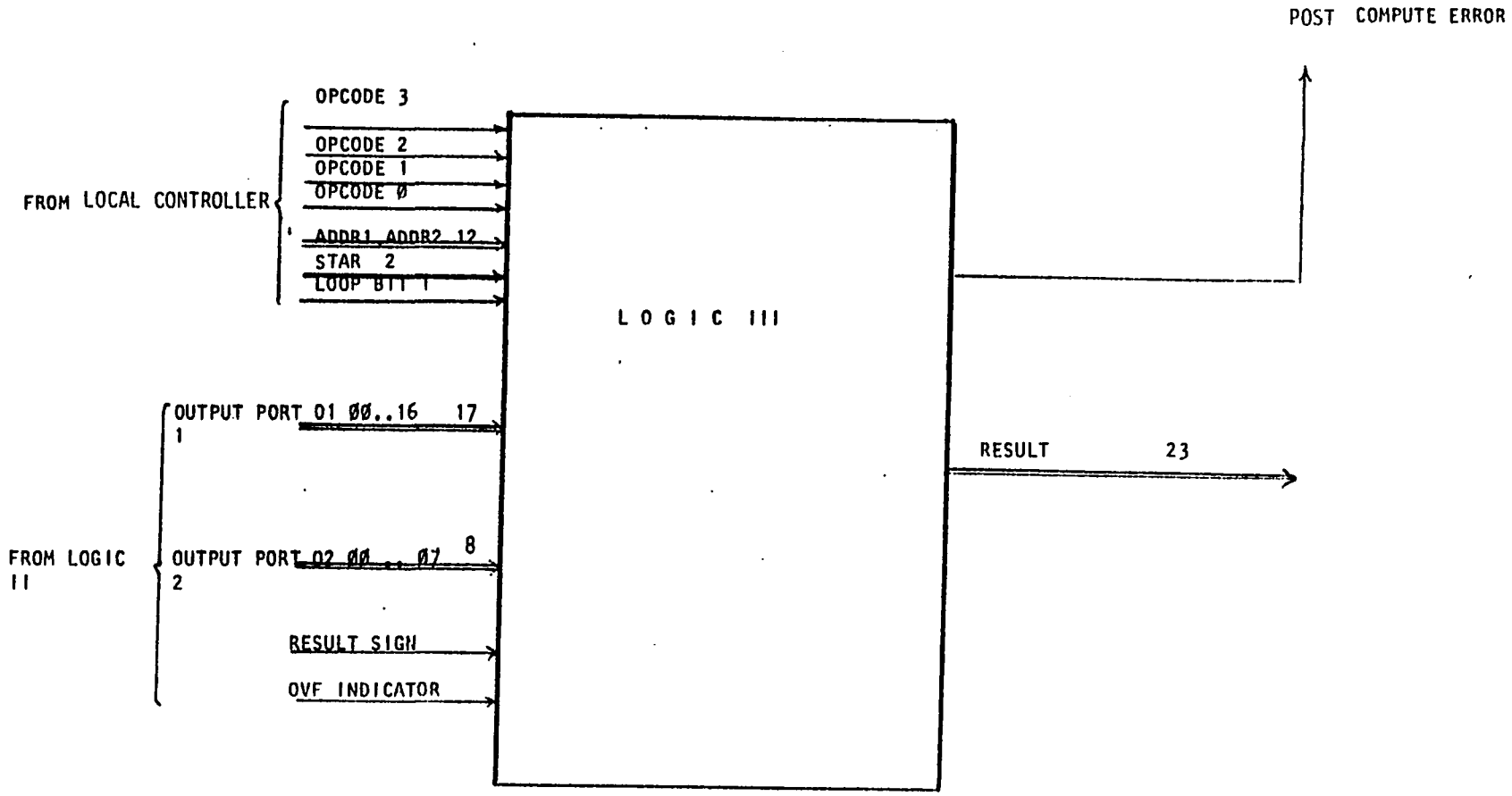


FIGURE 4.14:LOGIC III-INPUT/OUTPUT

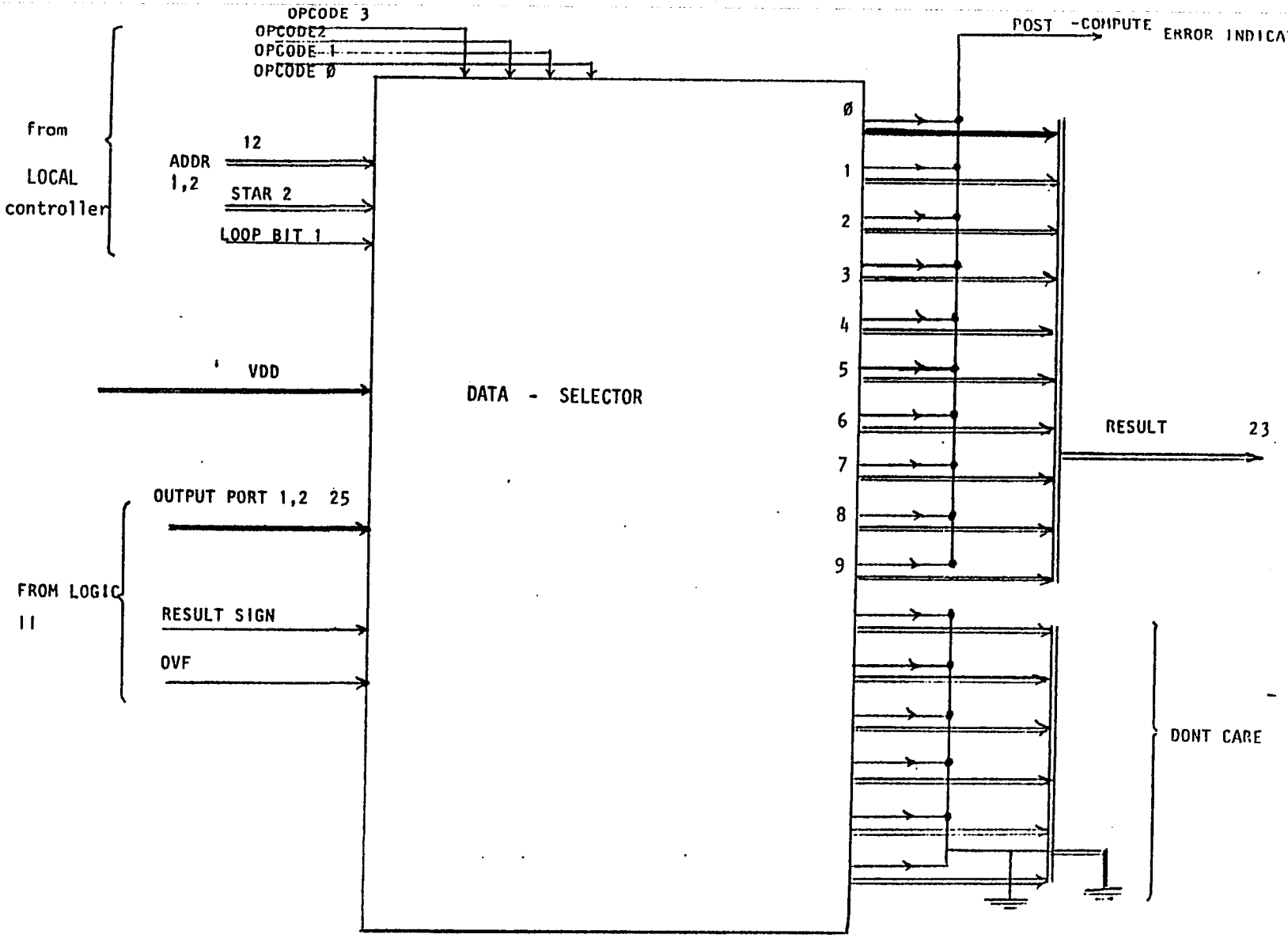


FIGURE 4.15:LOGIC III-HARDWARE BLOCK DIAGRAM

```
   Result (LSB):=O1 00;
   Result (2)  :=O1 02;
   Result (3)  :=O1 04;
   Result (4)  :=O1 06;
   Result (5)  :=O1 08;
   Result (6)  :=O1 10;
   Result (7)  :=O1 12;
   Result (MSB):=O1 14;
   Result (9)  :=result sign;

   Address 1:=Address 1;(*transmitted as it is*)
   Address 2:=Address 2;
   Star:=Star;
   Error indicator:=OVF;
   End;
If opcode =02h then (*subtract*)
   Begin
   Result (LSB):=O1 00;
   Result (2)  :=O1 02;
   Result (3)  :=O1 04;
   Result (4)  :=O1 06;
   Result (5)  :=O1 08;
   Result (6)  :=O1 10;
   Result (7)  :=O1 12;
   Result (MSB):=O1 14;
   Result (9)  :=result sign;

   Address 1:=Address 1;(*transmitted as it is*)
```

```
   Address 2:=Address 2;
   Star:=Star;
   Error indicator:=OVF;
   End;
If opcode =03h then (*multiply*)
   Begin
   Result (LSB):=O1 08;
   Result (2)  :=O1 09;
   Result (3)  :=O1 10;
   Result (4)  :=O1 11;
   Result (5)  :=O1 12;
   Result (6)  :=O1 13;
   Result (7)  :=O1 14;
   Result (MSB):=O1 15;
   Result (9)  :=result sign;

   Address 1:=Address 1;
   Address 2:=Address 2;  (*transmitted as it is*)
   Star:=Star;  (*transmitted as it is*)
     If (O1 16) true then
        error indicator:=1;
     Else
        error indicator:=0;
   End;
If opcode= 04h then (*divide*)
   Begin
   Result (LSB):=O2 00;
   Result (2)  :=O2 01;
```

```
   Result (3)  :=O2 02;
   Result (4)  :=O2 03;
   Result (5)  :=O2 04;
   Result (6)  :=O2 05;
   Result (7)  :=O2 06;
   Result (MSB):=O2 07;
   Result (9)  :=result sign;

   Address 1:=Address 1;
   Address 2:=Address 2;
   Star:=Star;
   Error indicator:=0;(*no post-compute error*)
   End;
If opcode= 05h then (*square-root*)
   Begin
   Result (LSB):=O2 00;
   Result (2)  :=O2 01;
   Result (3)  :=O2 02;
   Result (4)  :=O2 03;
   Result (5)  :=O2 04;
   Result (6)  :=O2 05;
   Result (7)  :=O2 06;
   Result (MSB):=O2 07;
   Result (9)  :=result sign;

   Address 1:=Address 1;
   Address 2:=Address 2;
   Star:=Star;
```

```
   Error indicator:=0;(*no post-compute error*)
   End;
If opcode =06h then (*square*)
   Begin
   Result (LSB):=O1 00;
   Result (2)  :=O1 01;
   Result (3)  :=O1 02;
   Result (4)  :=O1 03;
   Result (5)  :=O1 04;
   Result (6)  :=O1 05;
   Result (7)  :=O1 06;
   Result (MSB):=O1 07;
   Result (9)  :=result sign;
   Address 1:=Address 1;
   Address 2:=Address 2;
   Star:=Star;

   If (O1 08) true then
      Error indicator:=1;
   Else
      Error indicator:=0;
   End;
If opcode =07h then (*compare*)
   Begin (*generating operand 1 for the loop opcode*)
   If(O1 00)or(O1 02)or(O1 04)or(O1 06)or(O1 08)or(O1 10)or(O1 12)
   or(O1 14) false then
      Result(1,2,3,4,5,6,7,8,9):=000000000;
   Else
```

```
      result(1,2,3,4,5,6,7,8,9):=000000001;
   End;
If opcode =08h then (*loop*)
   Begin
   Result (LSB):=O1 00;
   Result (2)  :=O1 02;
   Result (3)  :=O1 04;
   Result (4)  :=O1 06;
   Result (5)  :=O1 08;
   Result (6)  :=O1 10;
   Result (7)  :=O1 12;
   Result (MSB):=O1 14;
   Result (9)  :=result sign;
   Address 1:= Address 1;
   Address 2:=Address 2;
   If (loop bit) true then
      Star:=11b;
   Else
      Star:=10b;
   Error indicator:=OVF;
   End;
If opcode= 09 h then(*dont care condition*)
   Begin
   Result(1.2.3.4.5.6.7.8.9):=000000000B;
   Address 1:=000000B;
   Address 2:=000000B;
   Star:=00B;
```

End;

END CASE.

4.10 LOCAL CONTROLLER

The local controller is the first stage in the processing element pipeline (Figure 4.1). The local controller has the following functions:

a) The local controller inputs the operands received from the instruction field into Logic I. It also inputs the control parameters to initialize logic I for particular opcodes.

b) The local controller generates the two bits; M bit and the LOOP bit.

   i) M bit:This bit is set to logic-0 for the add operation. For the subtract operation, it is set to logic-1. The M bit is used by Logic II.

   ii) LOOP bit: The loop bit is used for the loop opcode. The loop bit is a dont care for any other opcode and the local controller unconditionally sets the loop bit to logic-0. For the case of a loop opcode, the loop bit will be set to logic-0 by the local controller if Operand 1 is found to be an all zero vector. If operand 1 is found not to be an all zero vector, then the loop bit will be set to logic-1 by the local controller. The loop bit is used by Logic III.

c) The local controller inputs the following bits received from the instruction field.

   1)Opcode bits (4-bits)

2)Sign bits of Operand A/1 and Operand B/2.

3)Address 1(6-bit), Address 2 (6-bit) and Star bits(2-bit).

These bits are used by Logic II and Logic III. They are unchanged by the local controller.

d) The local controller checks for the pre-compute error prior to inputting the operands into the Logic I stages. If such a pre-compute error is in the operands, the master controller is notified in order to abandon all operations. The following errors are checked by the local controller.

1)Divide overflow error:

When the dividend (operand 1) is twice as long as the divisor (operand 2), then the condition of division overflow can be stated as :

A divide overflow condition occurs if the high order bits of the dividend constitute a number greater than or equal to the divisor [21].

The local controller checks for the four most significant bits of operand 2 to be zero (not including the sign bits, i.e., bits 4,5,6,7.). This condition insures that operand 1 is twice as long as operand 2. See Figure 4.16 and 4.17. Once this condition is true, the four most significant bits of operand 1 are compared with the four least significant bits of operand 2 by putting the bits through the comparator.

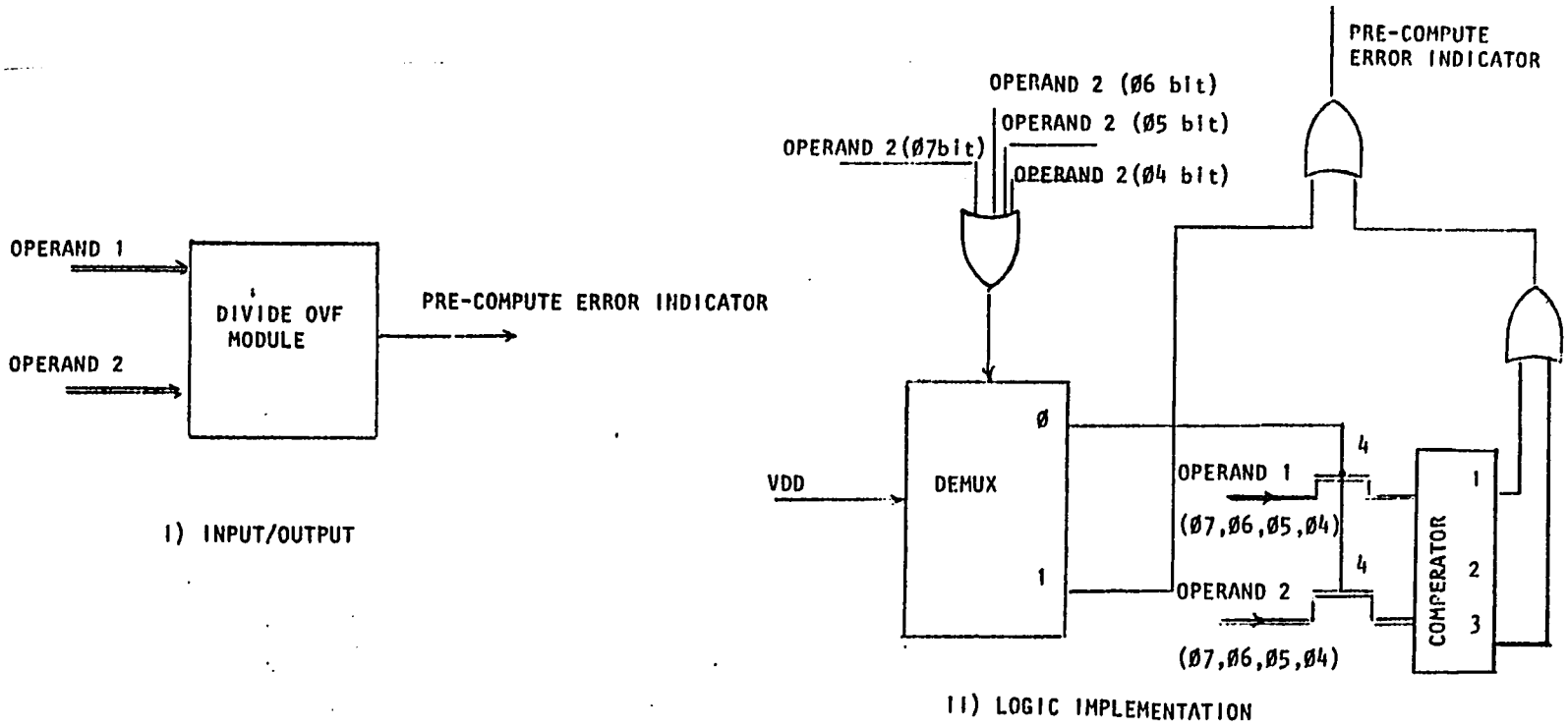


FIGURE 4.16:DIVIDE OVERFLOW IMPLEMENTATION

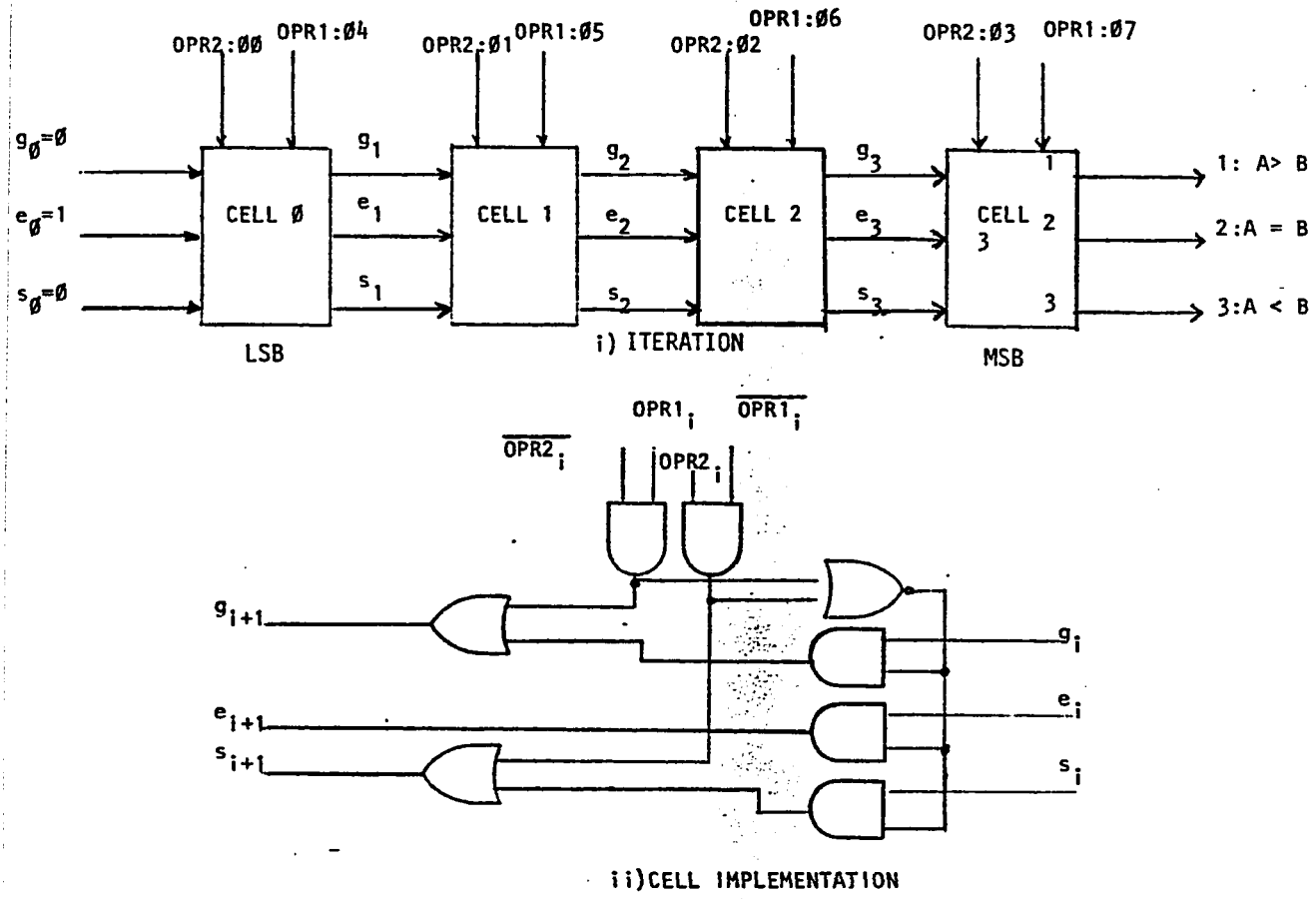


FIGURE 4.17:COMPARATOR LOGIC

The comparator design is shown in Figure 4.17[12]. The overflow logic mentioned above also indicates divide by error.

2)Negative square-root error:

The local controller inspects the sign bit of operand 1 if the opcode is found to be a square-root. A negative sign (bit 9=1) raises the pre-compute error indicator to logic-1.

e) For the COMP opcode, the local controller receives the two operands to be compared. The local controller generates the subtract opcode. A zero vector result out of Logic I indicates that Operand 1 is equal to Operand 2. Operand 1 does not equal Operand 2 if the result out of Logic I is a non-zero vector.

For the LOOP opcode, the local controller sets the loop bit according to Operand 1 received. It will input Operand 2 as the first operand and the zero vector (9-bit) as the second operand. It generates the new opcode as ADD. These operands and the opcode are put into the Logic I stages. The result out of Logic I is Operand 2; since zero is added to Operand 2.

Logic III inspects the loop-bit and conditionally sets the Star value to indicate either Address 1 or Address 2 as the valid destination address of the result, i.e., Operand 2.

The function of this setup is to put Operand 2 into the result bits and conditionally send it to either Address 1 or Address 2, depending on the loop bit generated by the previous COMP instruction.

4.11 DESCRIPTION OF LOCAL CONTROLLER LOGIC

INPUT

Instruction field consisting of:

1)Operand 1, Operand 2 (9-bits)

2)Opcode (4-bit)

3)Address 1, Address 2 (6-bits each)

4)Star (2-bit)

OUTPUT:

The output has three destinations:

a)Logic I.

1)Magnitude part of operand A/1 and operand B/2.

2)Control parameters.

b)Logic II and logic III.

1)Address 1, Address 2.

2)Star

3)M bit generated by the local controller.

4)Opcode.

5)Loop bit.

6)Sign operand A/1 and Sign operand B/2 transmitted by the local controller.

The input/output signals for the local controller are shown in Figure 4.18. Figure 4.19 gives the logic of the local controller. Address 1, Address 2 and the Star bits are input into the pipeline stages reaching Logic II and III, i.e., not going through the modules of logic I. The opcode bits from the received instruction are used as data-selectors. There are 10 opcodes and the rest of combinations are don't cares. These combinations are shown grounded (all-zero) in Figure 4.19.

An enable signal is emitted for only one of the 10 opcodes. This enable signal activates the DESCRIBED LOGIC which outputs the bits destined for Logic II and Logic III. In addition, the DESCRIBED LOGIC has pre-connections to Logic I according to the input format of Logic I. Once the operands are received into the DESCRIBED LOGIC by enable signal from the data-selector, the DESCRIBED LOGIC transmits the input operands with the pre-determined bit patterns (control parameters). Therefore, each enable signal (associated with each opcode) has a corresponding DESCRIBED LOGIC which physically connects the input operands to the Logic I stage.

The described logic is defined under the CASE of different opcodes.

4.12 DESCRIBED_LOGIC IN LOCAL CONTROLLER

```
CASE
If opcode =00H then
  Begin
  opcode:=01H;(*add*)
```

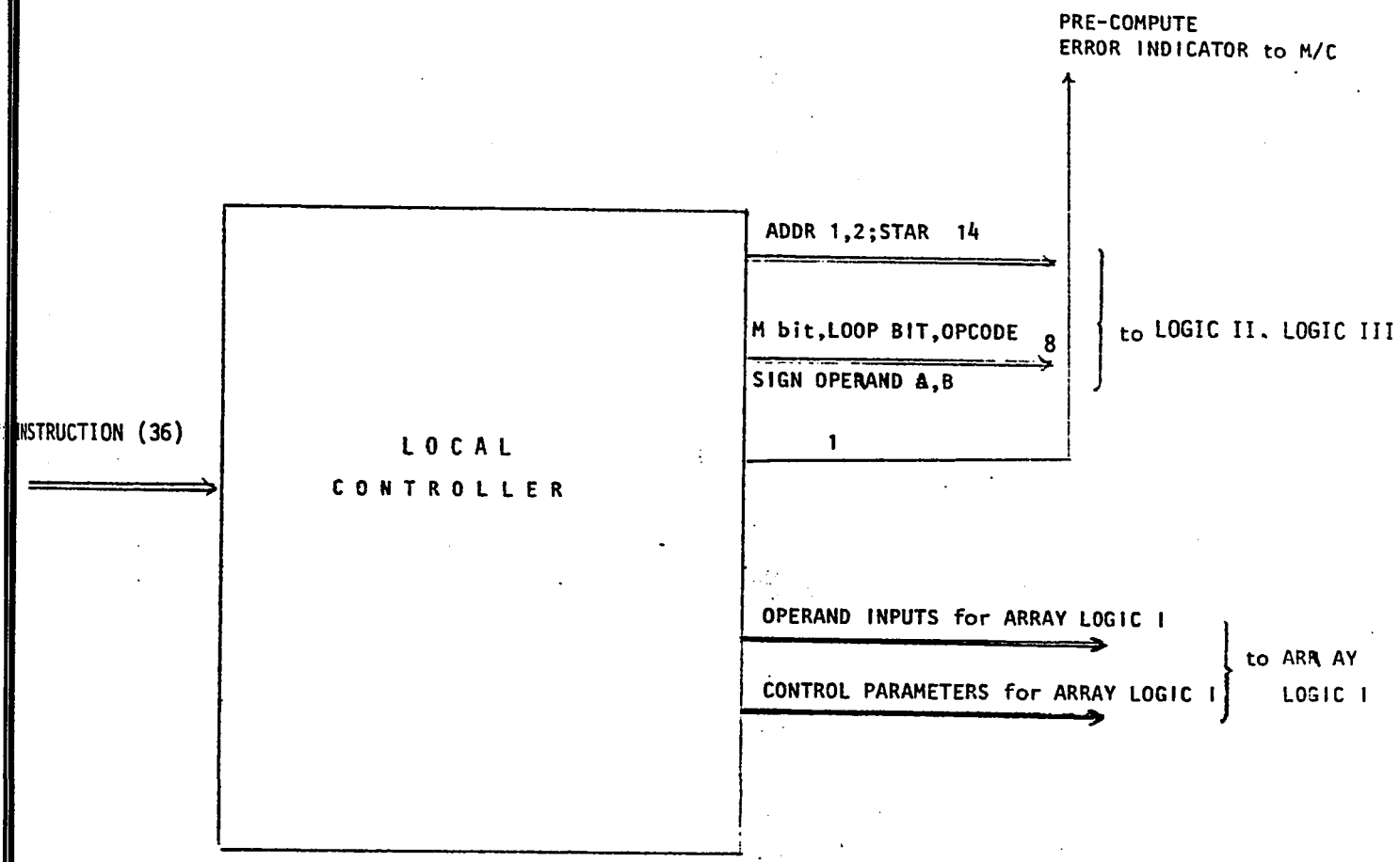


FIGURE 4.18:LOCAL CONTROLLER-INPUT/OUTPUT

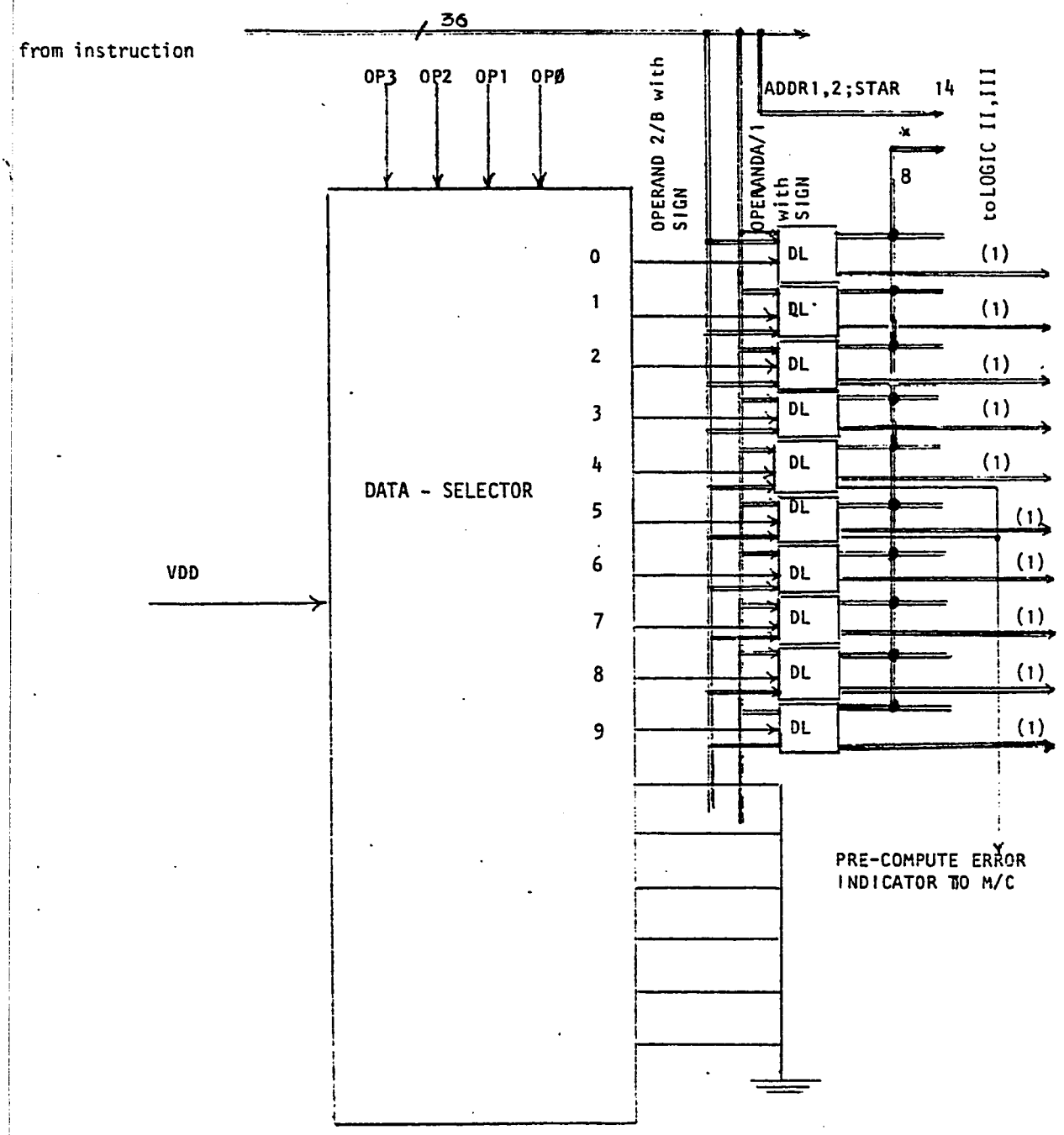


FIGURE 4.19:LOGIC BLOCK DIAGRAM OF LOCAL CONTROLLER

```
   loopbit:=0;
   M:=0;(*add*)
   Sign Operand A:=0;
   Sign Operand B:=0;

   (*CONTROL PARAMETERS*)
   Y:=1;
   Z(2i):=0; for 0<=i<=8
   Z(2i+1):=1; for 0<=i<=8
   Z(j):=0; for 18<=j<=28
   Z(2i):=0; for 0<=i<=8
   Z(2i+1):=1; for 0<=i<=8
   Z(j):=0; for 18<=j<=28

   ENX:=0;

   XGLB:=Sign Operand A ⊕ Sign Operand B ⊕ M

   (*OPERAND INPUTS*)
   A00=...=A28:=0;
   B00=...=B07:=0;
   B08=...=B15:=XGLB;
   C0=...=C15:=0;
   P0=...=P13:=0;
   End;
If opcode =01H then (*add*)
   Begin
   opcode:=01h;
   loopbit:=0;
```

```
   M:=0;
   Sign Operand A:=Sign Operand A;
   Sign Operand B:=Sign Operand B;

   (*CONTROL PARAMETER*)

   Y:=1;
   Z(2i):=0; for 0<=i<=8
   Z(2i+1):=1;for 0<=i<=8
   Z(j):=0; for 18<=j<=28
   Z(2i):=0;for 0<=i<=8
   Z(2i+1):=1;for 0<=i<=8
   Z(j):=0; for 18<=j<=28

   XGLB:=Sign Operand A @ Sign Operand B @ M

   (*OPERAND INPUTS*)
   A(2i):= ith bit of operand A, A(00) is LSB, for 0<=i<=7
   A(2i+1):= 0;for 0<=i<=6
   A15=...=A28:=0;
   B(i):= ith bit of operand B; for 0<=i<=7, B(00) is the LSB.
   B(08)=...=B(15):=XGLB;
   C00=...=C15:=0;
   P00=...=P13:=0;
   End;
If opcode =02H then (*subtract*)
   Begin
   opcode:=02H;
   loopbit:=0;
```

```
M:=1;(*subtract*)
Sign Operand A:=Sign Operand A;
Sign Operand B:=Sign Operand B;

(*CONTROL PARAMETER*)

Y:=1;
Z(2i):=0; for 0<=i<=8
Z(2i+1):=1;for 0<=i<=8
Z(j):=0; for 18<=j<=28
Z(2i):=0;for 0<=i<=8
Z(2i+1):=1;for 0<=i<=8
Z(j):=0; for 18<=j<=28

XGLB:=Sign Operand A ⊕ Sign Operand B ⊕ M

(*OPERAND INPUTS*)
A(2i):=ith bit of operand A, A(00) is LSB, for 0<=i<=7
A(2i+1):=0;for 0<=i<=6
A15...A28:=0;
B(i):=ith bit of operand B; for 0<=i<=7, B(00) is the LSB.
B(08)=...=B(15):=XGLB;
C00...C15:=0;
P00...P13:=0;
End;
If opcode =03H then (*multiply*)
Begin
opcode:=03H;
```

```
loopbit:=0;
M:=0;
Sign Operand 1:=Sign Operand 1;(*transmit*)
Sign Operand 2:=Sign Operand 2;(*transmit*)

(*CONTROL PARAMETERS*)

Y:=0;
Z:=0;
ENX:=0;
XGLB:=0;
(*OPERAND INPUT*)

P(i) = ith bit of operand 1, for 0<=i<=7, P(00) is LSB;
P08=...=P13:=0;
B00=...=B07:=0;
B(8+i):= ith bit of operand 2, for 0<=i<=7, B(08) is the LSB;
C00=B00, C01=B01,..., C15=B15;
A00=...=A28:=0;
If opcode =04H then (*divide)
Begin
opcode:=04H;
loopbit:=0;
M:=0;
Sign Operand 1:=Sign Operand 1;
Sign Operand 2:=Sign Operand 2;

Pre-compute Error:=Div.OVF(Operand 1, Operand 2);(*see Figure
4.16*)
```

```
(*CONTROL PARAMETERS*)

Y:=0;

Z:=0;

ENX:=1;

XGLB:=1;

(*OPERAND INPUT*)

A00=...=A07:=0;

A(8+j):=jth bit of operand 1, for 0<=j<=7, A(08) is LSB;

A16=...=A28:=0;

B00=...=B07:=0;

B(8+j):=jth bit of operand 2,for 0<=j<=7, B(08) is LSB;

C00=B00, C01=B01,...,C15=B15;

P00=...=P13:=0;

End;

If opcode =05H then (*square-root*)

Begin

opcode:=05H;

loopbit:=0;

M:=0;

Sign Operand 1:=Sign Operand 1;

Sign Operand 2:=0;

If Sign Operand 1 true then

  Pre-compute Error:=1;

Else

  Pre-compute Error:=0;
```

```
   (*CONTROL PARAMETER*)
   A(i):=i bit of operand 1, for 0<=i<=7, A(00) is LSB;
   A08=...=A28:=0;
   P00=...=P13:=0;
   B00=...=B13:=1;
   B14=B15:=0;
   C15:=0;
   C14:=1;
   C13=...=C00:=0;
   End;
If opcode =06H then (*square*)
   Begin
   opcode:=06H;
   loopbit:=0;
   M:=0;
   Sign Operand 1:=Sign Operand 1;
   Sign Operand 2:=0;(*dont care*)

   (*CONTROL PARAMETER*)

   Y:=0;
   Z:=0;
   ENX:=0;
   XGLB:=0;

   (*OPERAND INPUTS*)

   P(i) =ith bit of operand 1, 0<=i<=7, P(00) is LSB.
```

```
   P08=...=P13:=0;
   B00=...=B13:=1;
   B14=...=B15:=0;
   C15:=0;
   C14:=1;
   C13=...=C00:=0;
If opcode =07H then (*compare*)
   Begin
   opcode:=02H;
   loopbit:=0;
   M:=1;(*subtract*)
   Sign Operand A:=Sign Operand A;
   Sign Operand B:=Sign Operand B;

   (*CONTROL PARAMETER*)

   Y:=1;
   Z(2i):=0; for 0<=i<=8
   Z(2i+1):=1; for 0<=i<=8
   Z(j):=0; for 18<=j<=28
   Z(2i):=0; for 0<=i<=8
   Z(2i+1):=1; for 0<=i<=8
   Z(j):=0; for 18<=j<=28

   XGLB:=Sign Operand A @ Sign Operand B @ M

   (*OPERAND INPUTS*)
   A(2i):=ith bit of operand A, A(00) is LSB, for 0<=i<=7
```

```
   A(2i+1):=0; for 0<=i<=6
   A15=...=A28:=0;
   B(i):=ith bit of operand B; for 0<=i<=7, B(00) is the LSB.
   B(08)=...=B(15):=XGLB;
   C00=...=C15:=0;
   P00=...=P13:=0;
   End;
If opcode =08H then (*loop*)
   Begin
   Opcode:=01H;(*add*)
   M:=0;
   If (Operand 1=000000000B)true then
      loopbit:=0;
   Else
      loopbit:=1;
   Sign Operand A:=0;
   Sign Operand B:=Sign Operand 2;(*the operand which needs to be
   conditionally transfered*)

   Y:=1;
   Z(2i):=0; for 0<=i<=8
   Z(2i+1):=1; for 0<=i<=8
   Z(j):=0; for 18<=j<=28
   Z(2i):=0; for 0<=i<=8
   Z(2i+1):=1; for 0<=i<=8
   Z(j):=0; for 18<=j<=28

   XGLB:=Sign Operand A ⊕ Sign Operand B ⊕ M
```

(*Operand inputs*)

A(2i)=0; for 0<=i<=7, A(00) is the LSB;

A(2i+1):=0; for 0<=i<=6;

A15=...=A28:=0;

B(i)= ith bit of operand B; for 0<=i<=7, B(00) is LSB.

B(08)=...=B(15) :=XGLB;

C00=...=C15:=0;

P00=...=P13:=0;

End;

If opcode =09H then (*finish*)

Don't care (*this condition will not occur since the finish opcode will not reach the processing element-the fetch unit will indicate the completion of operation to the master controller *)

END CASE

4.13 DESCRIPTION OF MASTER CONTROLLER

The master controller is a finite state machine. In the DFEE system, the master controller acts as the control unit, monitoring the execution phase of the DFEE system. However, since all the sub-units of the DFEE execution unit form a circular pipeline, these sub-units require little control from the master controller.

The master controller also acts an an interface between the DFEE system and the host computer. The following protocol is followed

between the master controller and the host computer:

4.13.1 HOST COMPUTER AND DFEE INTERFACE THROUGH THE MASTER CONTROLLER

The memory of the host computer contains the instructions ready to be transfered to the cell blocks of the DFEE system. These instructions represent the data-flow graph computation. At each clock phase 1, the host computer should have a new instruction on its 36-bit DATA-IN lines (Figure 4.20). This instruction is pushed into the cell block instruction cell 1. During phase 2 the instruction is transfered one instruction cell down in the memory stack. In the loading phase, the memory cell blocks behave as a one-directional stack of length 30.

The master controller is signaled that the data is coming in through the DATA-IN lines by the host computer. The host computer enables the DATAIN line, a single bit flag to logic high. This is an indication to the master controller that valid instruction resides on the DATA-IN line, ready to go inside the memory cell blocks. The master controller responds by giving proper control signals to the DFEE execution unit so that the instructions can be loaded into the memory cell blocks. The master controller achieves this by making the START line active high, putting the LOAD signal

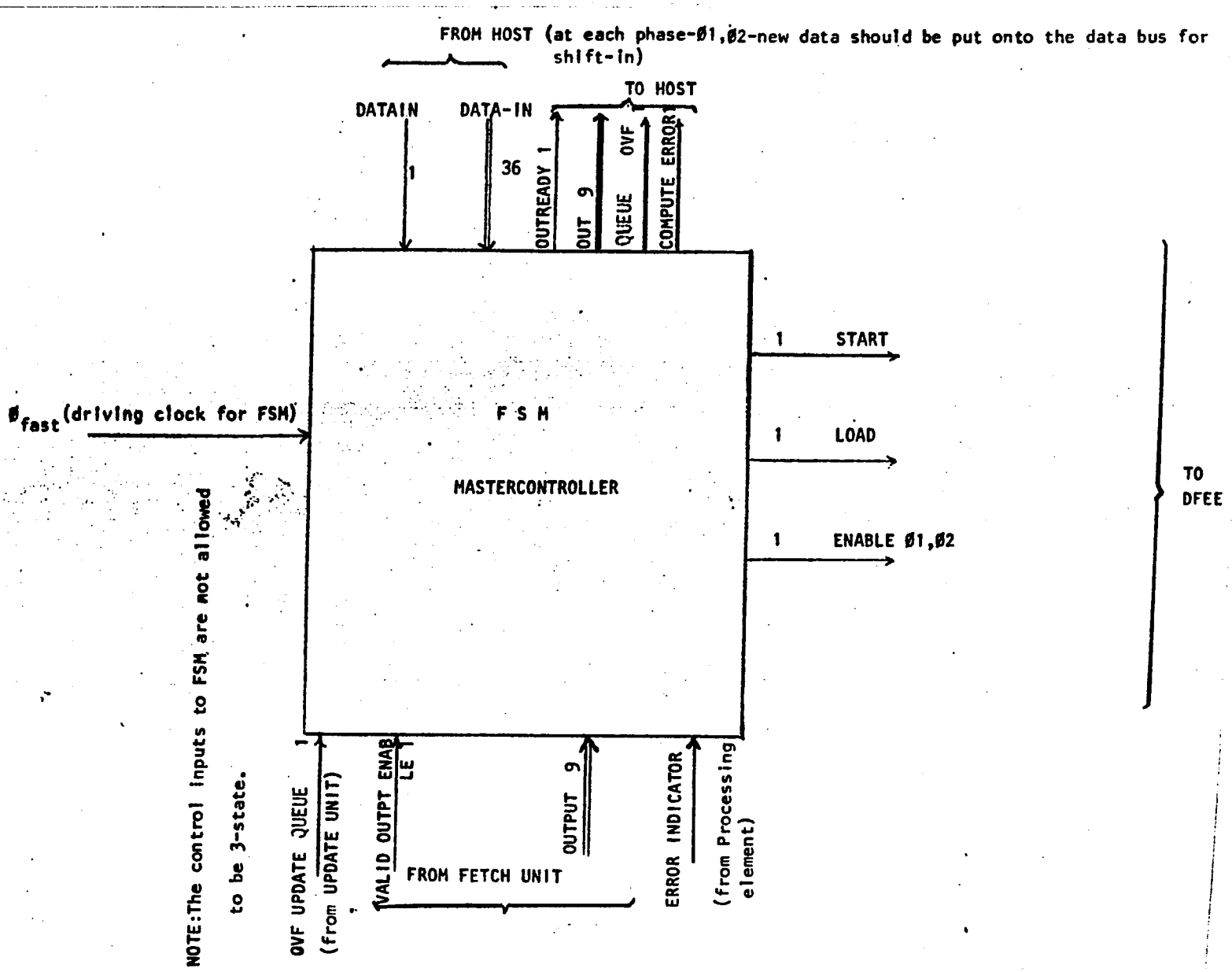


FIGURE 4.20:MASTER CONTROLLER-INPUT/OUTPUT

to high . The master controller also enables the phi 1 and phi 2 to the DFEE execution unit (Figure 4.20). As long as the START signal is active high, it insures that the sub-units of the DFEE system cannot interfere in the loading phase.

During the loading phase, the DATAIN line is active high. The DATAIN line goes low under the host computer command when the host computer has given out all the instructions from its memory into the cell block memory of the DFEE system. Once this is done, the DATAIN line goes low and the master controller puts the DFEE system into execution phase.

The execution phase begins when the master controller makes the START and LOAD signals low. During the execution phase the master controller checks for two types of errors:

a) Update queue overflow

This error indicates that the instruction addresses to be communicated to the fetch unit have exceeded two in the same phase, i.e., update queue overflow. This is indicated to the master controller by the update unit through the 1-bit signal OVF UPDATE. The master controller indicates this to the host computer by the one bit signal, QUEUE OVF. Further, the master controller abandons all operations in the DFEE execution unit by disabling the clocks phi 1 and phi 2, putting the START signal active high and resetting the DFEE system by grounding the data-paths in the circular pipeline (Figure 3.3).

The QUEUE OVF signal points to the host computer that the software responsible (discussed in Chapter 5; Optimization Compiler)for generating the instructions from the data-flow graph has failed in placing the instructions in the correct form; i.e., it did not satisfy the constraints.

b) Computation error

While the DFEE is in execution phase, the master controller also checks for the computation errors, e.g., divide-by-zero, square-root of a negative number, overflow. If such an error is indicated from the processing elements by the single bit line ERROR INDICATOR, the master controller abandons the operations in the DFEE by disabling the phi 1, phi 2 and putting the start signal to active high. Further, it issues a strobe active high shown as COMPUTE ERROR in Figure 4.20, to the host computer.

This signal indicates to the host computer that the computation has failed and that the host computer may reconfigure the data-flow graph by writing the computation in another way and carrying out the procedure again. For example, a divide operation before multiplication may save the overflow error.

If the computation is carried out successfully, then the fetch unit finally finds the FIN opcode in one of the instruction cells. The fetch unit obtains the output in Operand 1 sub-field of the instruction cell and puts the output on the 9-bit OUTPUT lines to the master controller.

The fetch unit also puts a single bit flag, VALID OUTPUT ENABLE to logic high. As soon as the master controller receives this input, it enables the connection of these 9-bit lines to the OUT lines going to the host computer. In order to indicate to the host computer that the valid output is available, the master controller puts the OUTPUTREADY signal to the host computer active high.

The master controller waits for another computation to be performed by waiting for the DATAIN line to be active high again. Once, in this state, the master controller resets the DFEE system, either with DATAIN line active high or low, thus making sure that all the system data-paths are resetted for the next computation. But the OUT lines carrying the output of the computation are not resetted (Figure 4.20).

4.13.2 ALGORITHMIC STATE MACHINE DESCRIPTION OF THE MASTER CONTROLLER

MASTER CONTROLLER

INPUT: DATA-IN transfer lines, OVF UPDATE QUEUE,
VALID OUTPUT ENABLE, OUTPUT transfer lines,
ERROR INDICATOR;

OUTPUT:START,ENABLE phi 1-phi 2,LOAD,OUTPUTREADY,
OUT transfer lines, COMPUTE ERROR, QUEUE OVF);

1. IF DATAIN high THEN

   a.START high.

   b.LOAD high.

   c.ENABLE system clock phi 1/Phi 2 (*loading phase*).

   d.Transfer DATA-IN into cell block memory.

   e.Go to 2.(*check for loading complete*)

ELSE Go to 1.

2. IF DATAIN low THEN

   a.START low.(*loading complete*)

   b.LOAD low.

   c.ENABLE system clock phi 1/phi 2.

   d.Go to 3.

ELSE

   a.Transfer DATA-IN into cell block memory.(*still loading phase*)

   b.Go to 2.

3. IF VALID OUTPUT ENABLE high THEN

   a.OUTPUTREADY high.(*Execution phase*)

   b.Transfer OUTPUT to OUT for the host computer.

   c.Go to 1.

ELSE Go to 4.

4. IF OVF UPDATE QUEUE true THEN

   a.START high.

   b.DISABLE system clock phi 1/phi 2.

   c.QUEUE OVF high.

   d.Go to 1.

ELSE Go to 5.

5. IF ERROR INDICATOR true THEN

a.START high.

b.DISABLE system clock phi 1/phi 2.

c.COMPUTE ERROR high.

d.Go to 1.

ELSE Go to 3.

END MASTER CONTROLLER

4.13.3 HARDWARE IMPLEMENTATION OF THE MASTER CONTROLLER

The ASM (Algorithmic state machine) state diagram describing the master controller is given in Figure 4.21.

The master controller is implemented by a PLA (programmable logic array) structure. The design of the master controller is such that a PLA of reduced size is obtained. Each micro-instruction of the master controller is to be executed during phi 1 / phi 2, i.e., the master controller is implemented as a microprogrammed controller. The execution of the micro-instructions need to be synchronized with a faster clock. This clock has been called ϕ_{fast}.

Table I shows the ASM state table for the corresponding Figure 4.21. There are six micro-instructions to be executed during phi 1/phi 2. Therefore

$\phi_{fast} \simeq 10 \times \phi 1/2$

The OUTPUTREADY flag from the master controller will be high during only one period of ϕ_{fast} and in the next period, it will be resetted for the next computation. Therefore, the host computer must detect this flag during one period of ϕ_{fast} (Figure 4.21).

The master controller is implemented in an AND-OR plane configuration with complementary registers separating the two planes. The complete implementation of the master controller is shown in Figure 4.22. In order to force the 00 state at power-on, the power on signal is used which would set the master controller from any invalid initial state to state 00(see Figure 4.20).

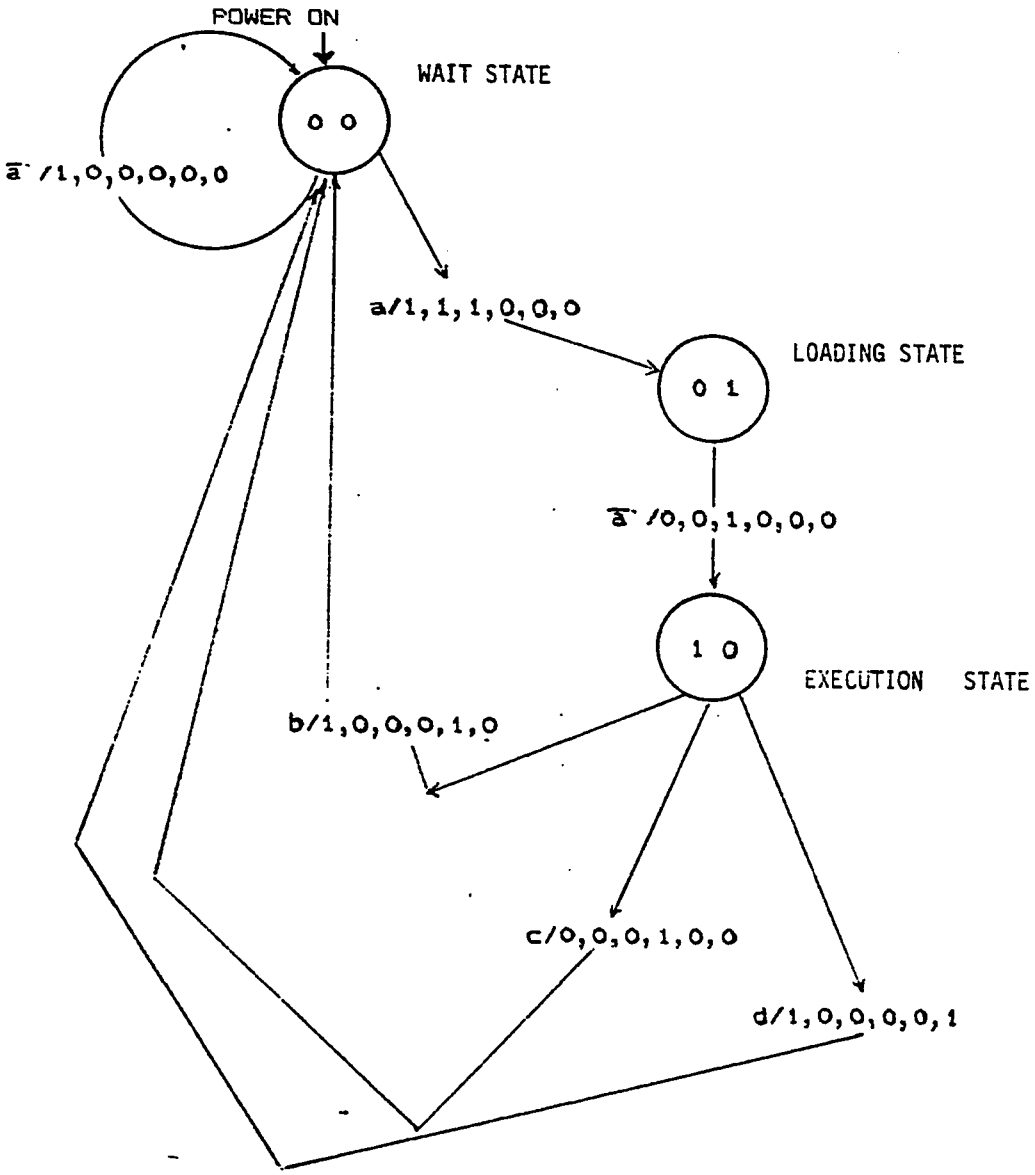


FIGURE 4.21:STATE DIAGRAM

STORED DURING Øfast						STORED DURING Ø fast							
INPUTS				PRESENT STATE		NEXT STATE		OUTPUTS					
a	b	c	d	Y1	YO	Y1	YO	A	B	C	D	E	F
0	X	X	X	0	0	0	0	1	0	0	0	0	0
1	X	X	X	0	0	0	1	1	1	1	0	0	0
0	X	X	X	0	1	1	0	0	0	1	0	0	0
X	1	X	X	1	0	0	0	1	0	0	0	1	0
X	X	1	X	1	0	0	0	0	0	0	1	0	0
X	X	X	1	1	0	0	0	1	0	0	0	0	1

CODING:

INPUTS: a:=DATAIN

b:-OVF UPDATE QUEUE

c:-VALID OUTPUT ENABLE

d:-ERROR INDICATOR

OUTPUTS: A:- START

B:- LOAD

C:- ENABLE Ø1,Ø2

D:- OUTPT REDY

E:- QUEUE OVF

F:- COMPUTE ERROR

TABLE I

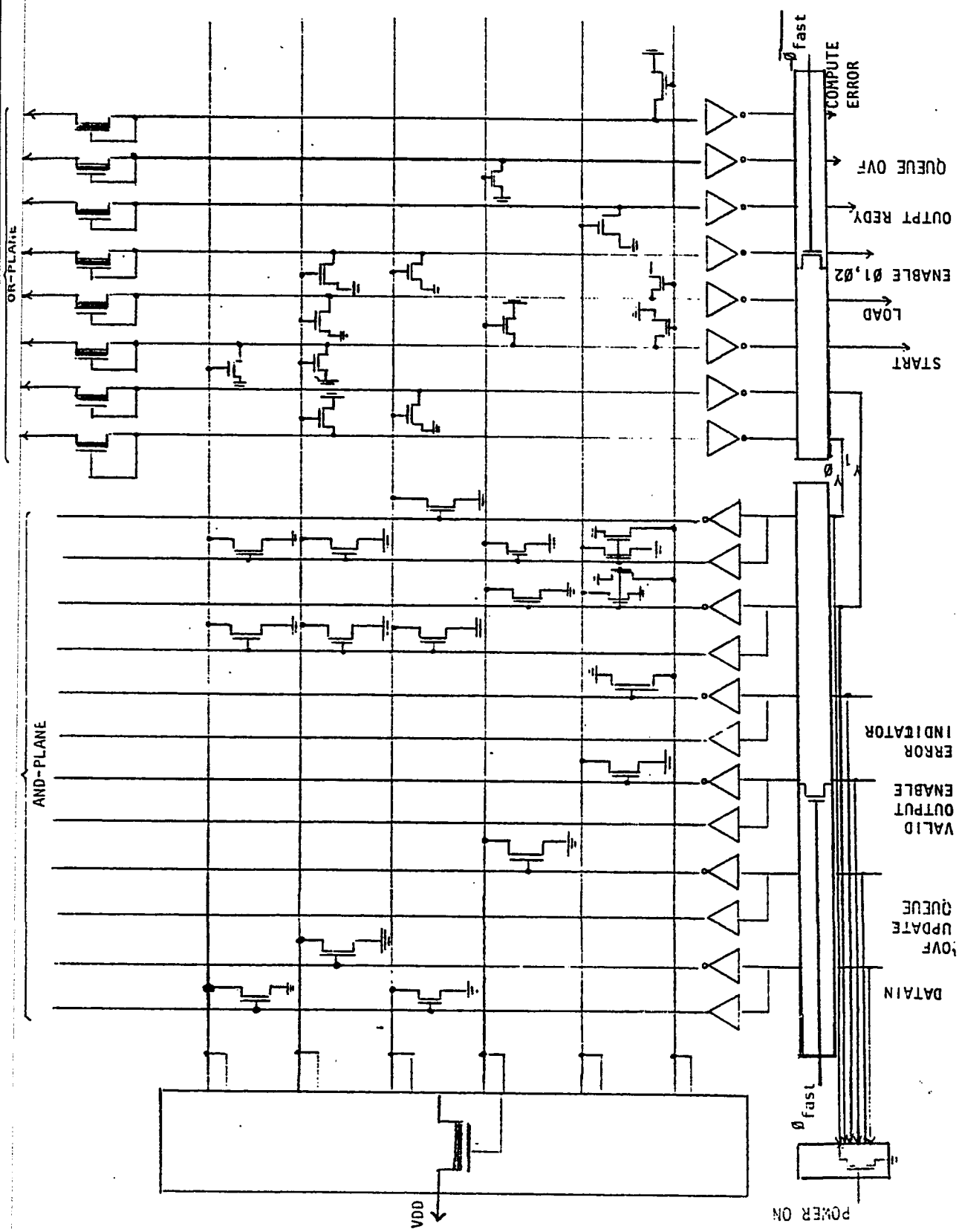


FIGURE 4.22: IMPLEMENTATION OF MASTER CONTROLLER

CHAPTER 5

OPTIMIZATION COMPILER

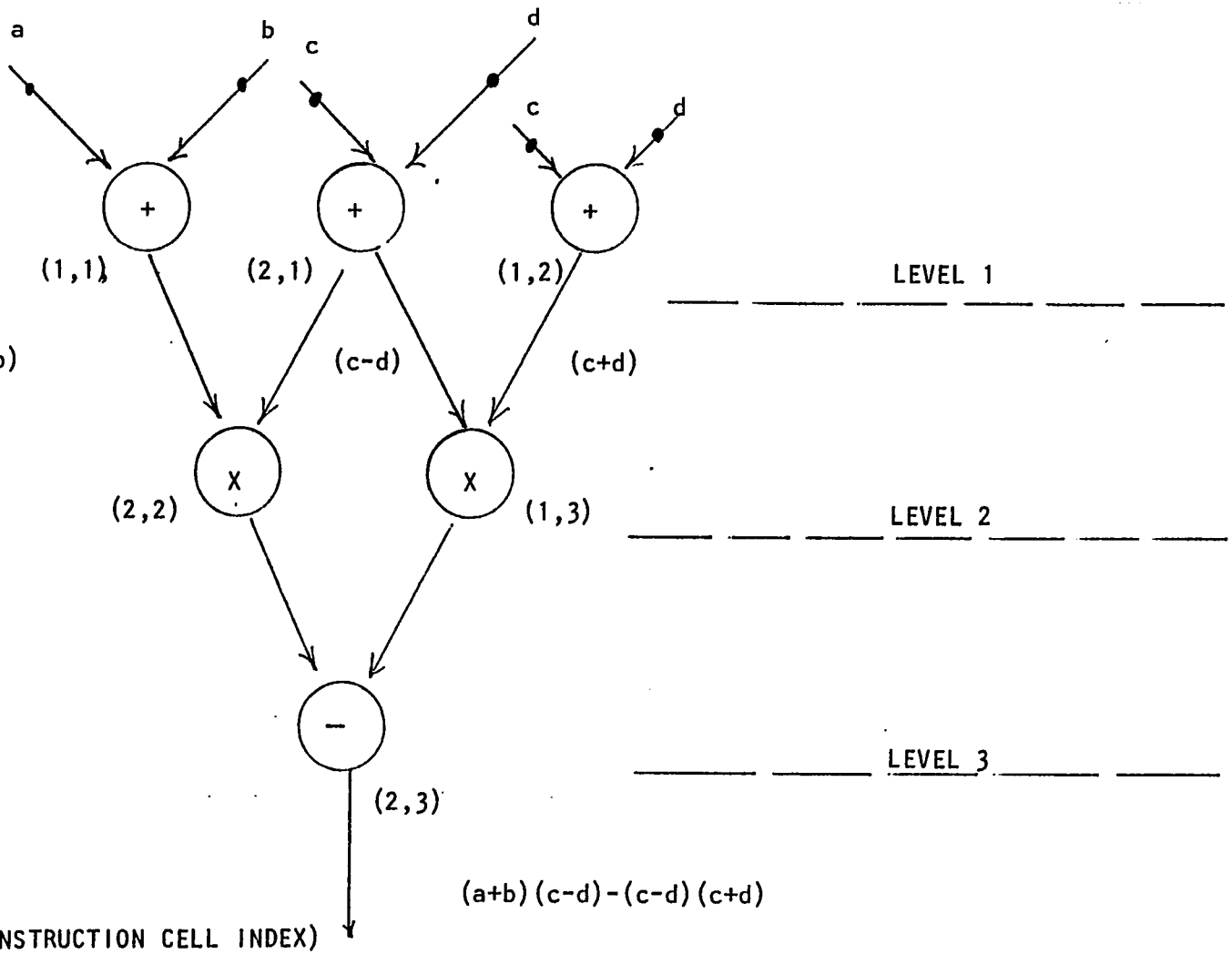
5.1 FUNCTIONAL DEFINITION

There are two tasks the optimization compiler must do:

a) The optimization compiler generates the instructions representing the data-flow graph computation from the expression input and compiles these instructions into the memory cell blocks of the DFEE system.

b) Consider the data-flow graph computation of Figure 5.1. The data-flow graph can be conveniently divided into levels. Each level represents the depth of the data-flow graph. Each level i computation must be done before the level i+1 computation. Thus the three operations (+,-,-) need to be executed first, since the data-tokens are present at the input arcs. These instructions are full, i.e., executable. Three instructions are formed from level 1. Level 2 instructions will wait for the data-tokens from level 1. Similarly, the single instruction in level 3 will wait for the data token from level 2. Each circle (node) represents an instruction in the data-flow graph.

Next, consider the allocation of instruction cells for these instructions. The architecture contains two memory cell blocks, each containing fifteen instruction cells. Two instruction cells



Note:(i,j)=(CELL BLOCK INDEX,INSTRUCTION CELL INDEX)

FIGURE 5.1

can be executed concurrently through the two fetch unit-processing element pair.

If the level 1 instructions are put in the same Cell Block I and level 2 , level 3 instructions in Cell Block II, then only one instruction will be executed during the first execution phase. This is so because level 2 and level 3 instructions in Cell Block II will be waiting for the data-tokens from Cell Block I. In order to execute two instructions in each execution phase, the instructions should be allocated accordingly. The key to this allocation is the level index.

Cell Block I and Cell Block II are filled with level 1 instructions until all the instructions are filled in the remaining instruction cells of cell block I and cell block II. Next, level 2 instructions are filled in the remaining instruction cells of cell blocks I and II. Level 3 instructions are allocated in the same fashion.

All level i instructions are allocated in Cell blocks I and II. The process continues until all the levels have been finished. If there are n levels and each level contains IP(j) instructions, then the following algorithm puts the instructions alternatively into Cell block I and II.

```
BEGIN ALLOCATE
cell block index:=1;
For i:=1 to n do (*level index*)
```

```
BEGIN
For j:=1 to IP(i) do (*IP is the number of
                      instructions in level i*)
  BEGIN
  1.allocate instructions(j) in cell block(cell block index);
  2.cell block index:=cell block index +1;
  3.if cell block index >2 then cell block index:=1;
  END;
END;
END ALLOCATE
```

This allocation rule utilizes the full parallelism of the architecture, i.e., the two processing elements. Figure 5.1 is marked with cell block index and instruction cell index (i,j) under each circle representing the instruction.

5.2 USER VIEW OF DATA-FLOW EXPRESSION EVALUATOR SYSTEM

The user's view of the DFEE system is as follows:

i) For the straight expression, the user will access the DFEE system in the following way;

a:=(b-c) x (d-b) + b

where a, b, c and d represent sign-magnitude integers.

OPTCOMPILER(INPUT: exp[(b-c)x(d-b) + b]; OUTPUT:a)

On return from the procedure OPTCOMPILER, the user will

have a value in the variable a.

ii) For the case of iteration/loop expression:

y(i+1):=(a-b) x (y(i)-d);

For i:=initial value to final value;

The user will access the DFEE in the following way:

OPTCOMPILER(INPUT: loop([(a-b)x(y-d)]; y=y (initial value); initial value := positive integer;final value := positive integer; OUTPUT:y);

At the return from the routine OPTCOMPILER, the output would be in variable y. The iteration could be from a positive integer to another positive integer with the condition:

final value > initial value

Y(initial) is an integer which can be either positive or negative.

The user cannot evaluate an iterative expression which results in a data-flow graph of the type shown in Figure 5.2. The feedback node cannot have more than one input which is to be updated from its predecessor instruction. If the feedback node instruction represents a unary operand instruction then this condition does not apply. If the feedback node instruction is a binary operand instruction, then one of the operands should always be a literal value/constant. Figure 5.3 shows a correct data-flow graph representing the loop schema.

The input to the optcompiler procedure will be a string denoting the expression. The input will distinguish the straight

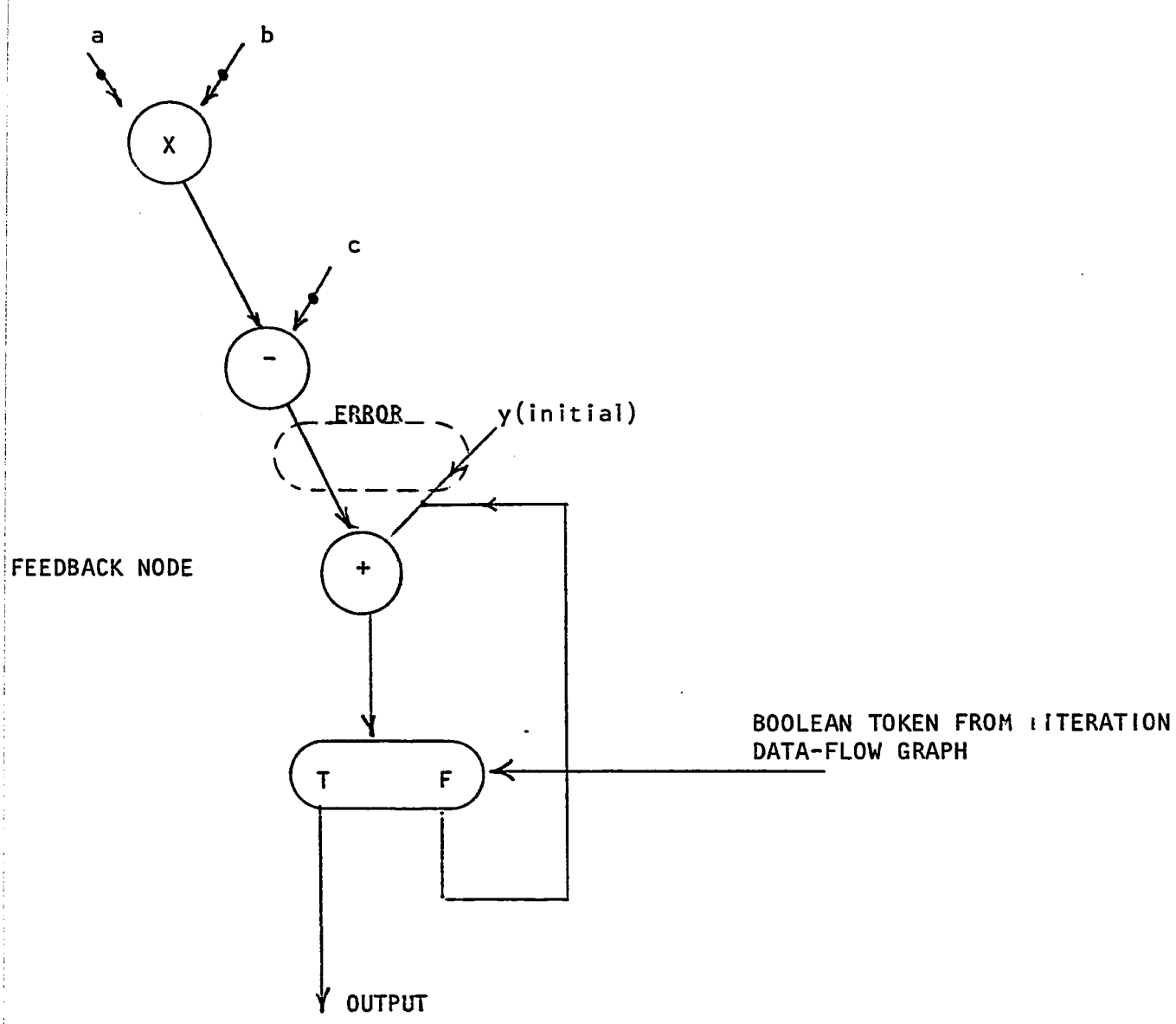


FIGURE 5.2: INCORRECT LOOP ITERATION SCHEMA

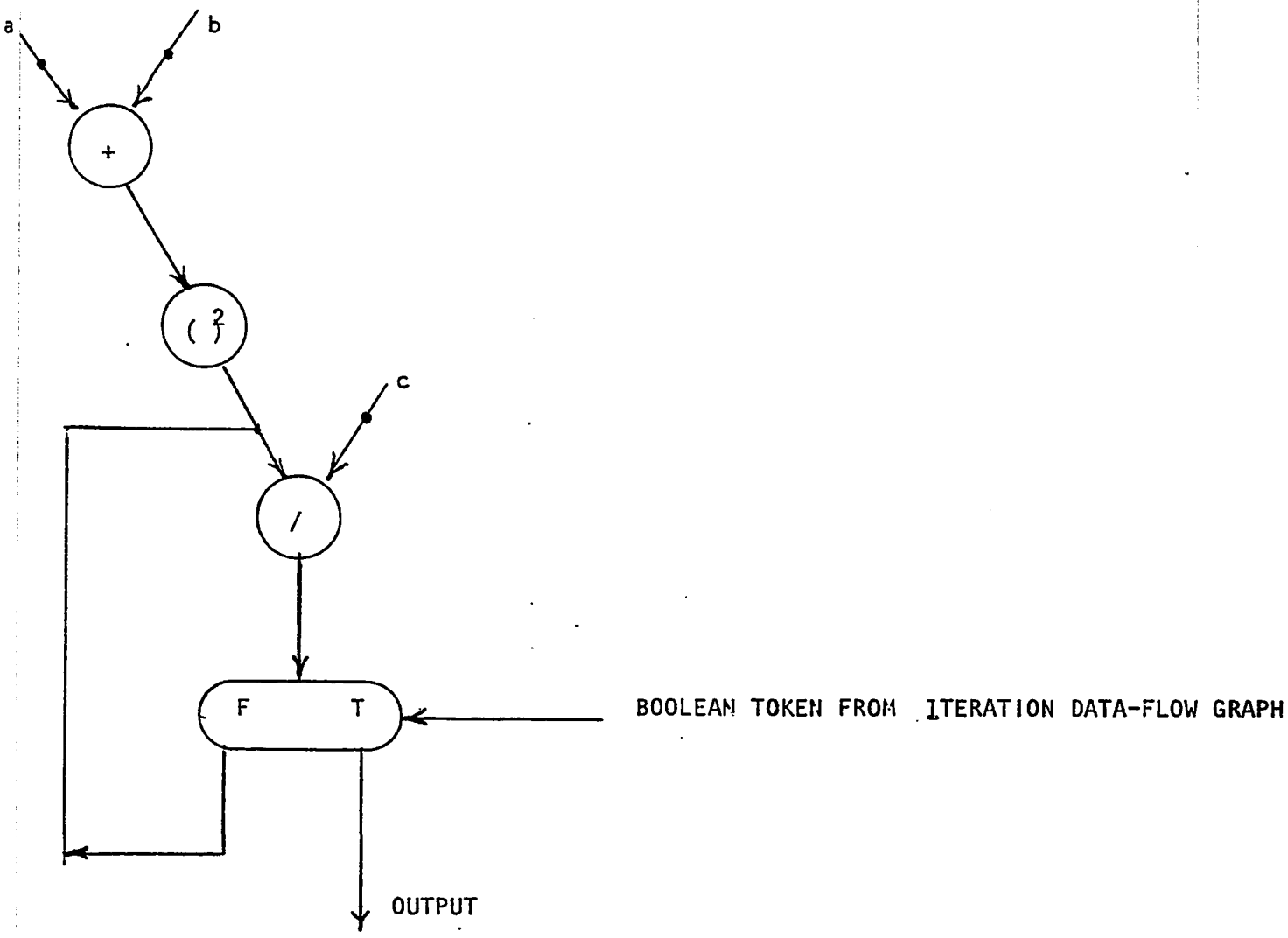


FIGURE 5.3:CORRECT FORM OF DATA-FLOW GRAPH FOR LOOP ITERATION SCHEMA

expression (exp) and the loop schema (loop) from the string form.

All the constants/literals will have a defined signed integer value in the memory of the host computer. Each literal will be input as a fixed-point signed-integer in the hardware of the DFEE system.

The output of the optcompiler will represent a single variable defined at the time of inputting the string expression. For example, in the case of straight expression it is 'a'.

At the return from the procedure optcompiler, the variable 'a' will contain the output value of the computation expression. The same argument applies for 'y', the loop schema output.

While the output is not available from the optcompiler, the host computer can do other tasks. The host computer will act as a multiprocessing system, since concurrent execution of processes on independent nodes of the host computer are able to exist [2]. One process of expression evaluation is going on in the DFEE system which is an independent node of the host computer. The other process can either be started or resumed by the host computer CPU. The optimization compiler and host computer interaction is shown in Figure 5.4.

5.3 REQUIREMENTS ON THE OPTIMIZATION COMPILER

The DFEE architecture imposes the following requirements on the optimization compiler:

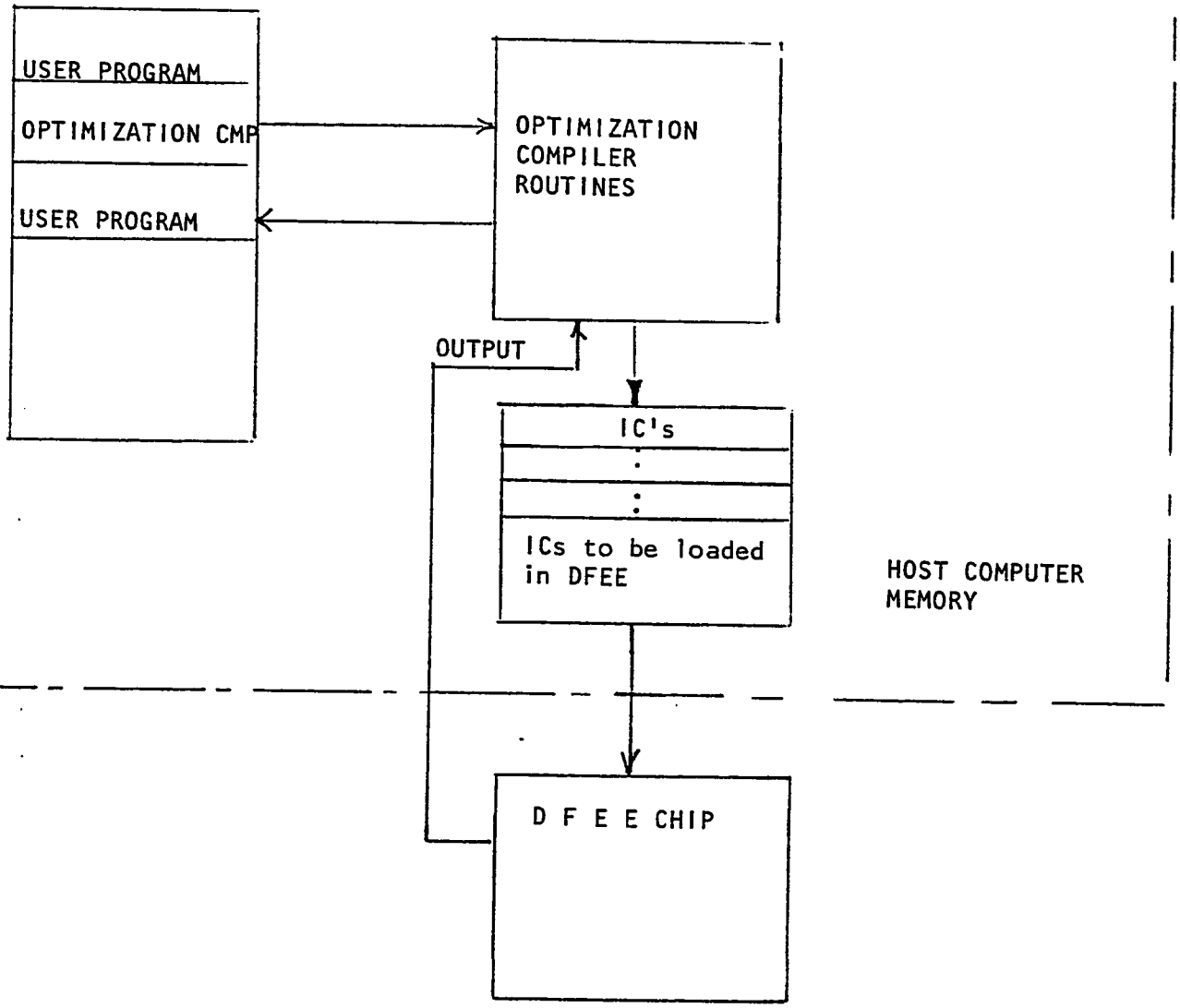


FIGURE 5.4: OPTIMIZATION COMPILER - HOST COMPUTER INTERACTION

1) Constraint satisfaction

a)The instructions in the cell blocks should be placed such that the IC Address Queue will be filled with two addresses simultaneously in the case of two updates , if and only if the IC Address Queue was empty in the previous cycle.

b)For the loop schema computation, the following constraint should be satisfied:

There are two data-flow graphs used to execute loop schema, one data-flow graph for the computation, the other data-flow graph for counting the iteration. There has to be synchronization between the boolean token passed between the the two data-flow graphs. Synchronization is established by inserting the null instructions as shown in Figure 2.9.

2) Blank instructions for filling the memory cell blocks

The instruction cells which are not used by the host computer should contain blank fields, i.e., operand fields should contain 100H while the opcode, Address 1, Address 2 and Star fields should contain logic-0 values. The optimization compiler generates these blank instructions to load all the instruction cells in the memory cell blocks of the DFEE system, i.e., 30 instruction cells.

3) Blank operands

In generating the instructions and loading the instruction

cells, the blank operand fields should contain 100H.

4) **Blank instructions for addressing more than two instructions**

Consider the data-flow graph representation in Figure 5.5. At level 1, the instruction with SUBTRACT opcode gives the result to three instructions in level 2. But three different addresses cannot be handled by the architecture of DFEE. The optimization compiler must insert more instructions into the data-flow graph to accomodate more than two addresses. These instructions do not modify the result to be transmitted but merely alter the addresses. Thus, if the result has to go to three next level instructions, one extra blank instruction can be inserted. If the result has to go to four addresses, two extra blank instructions can be inserted. Similarly, if the result has to go to six instructions in the next level, four extra blank instructions can be inserted. The modified data-flow graph for the data-flow graph shown in Figure 5.5 is shown in Figure 5.6. The extra blank instruction is distinguished by shade. However, this will be at the cost of extra execution cycles, since the extra instructions will go through the circular pipeline.

For some expressions, this can be avoided by inspection. Consider the following expression:

((a+b) x b)x(((a + (a+b))+(a+b))+((a+b)+(c-d)))

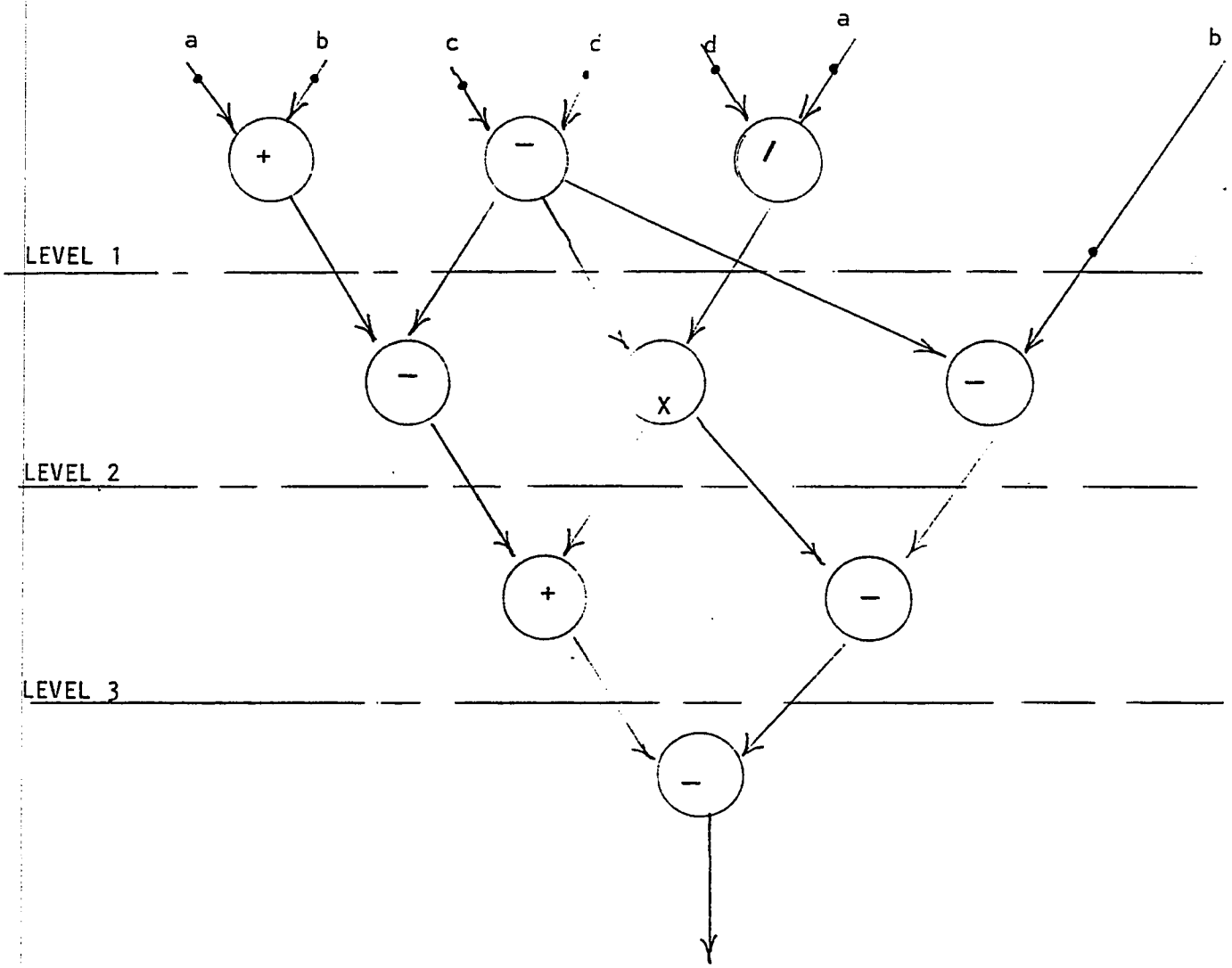


FIGURE 5.5

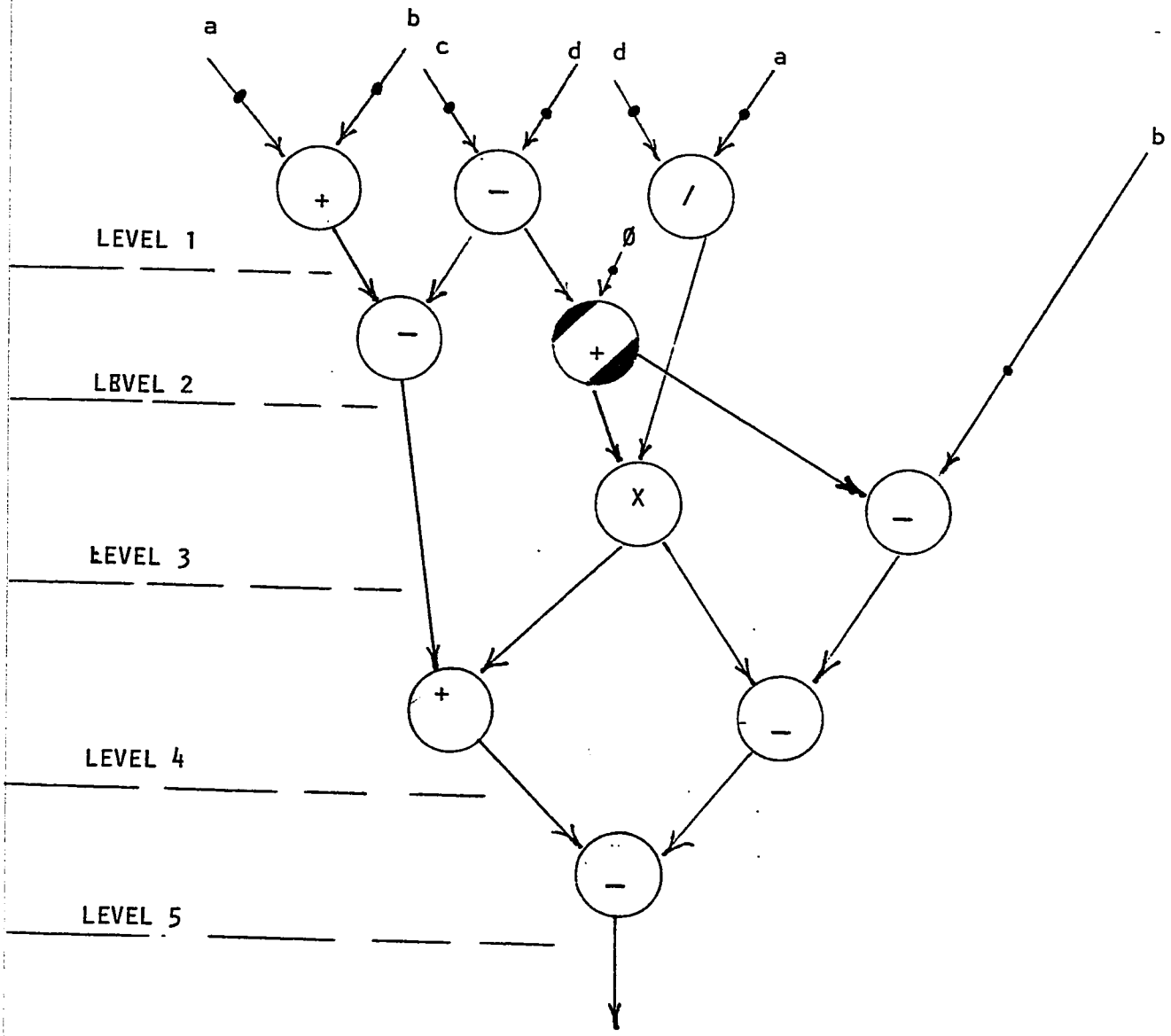


FIGURE 5.6

The data-flow graph for this expression is shown in Figure 5.7. There are three addresses required in transfering the result of (a+b) to level 2. Equivalently, the expression can be represented as:

((a+b)xb)x((2x(a+b))+(a+(c-d)))

The data-flow graph for this expression is shown in Figure 5.8. There are no three addresses required in this graph.

5) **Generation of FIN opcode for the termination of data-flow graph computation**

In addition to generating the instructions from the data-flow graph, the optimization compiler must generate the final FIN opcode instruction. This instruction contains the final result of the computation. The predecessor instruction computes the final result of the computation. The final result is routed into Operand 1 field of the FIN opcode instruction. The fetch unit detects the termination of the computation by the FIN opcode instruction.

6) **Instruction formation for the loading phase**

In the loading phase, the instructions would be loaded into the DFEE memory as in the case of stack loading. The instruction cells which would be empty will contain blank instructions. The optimization compiler will generate the

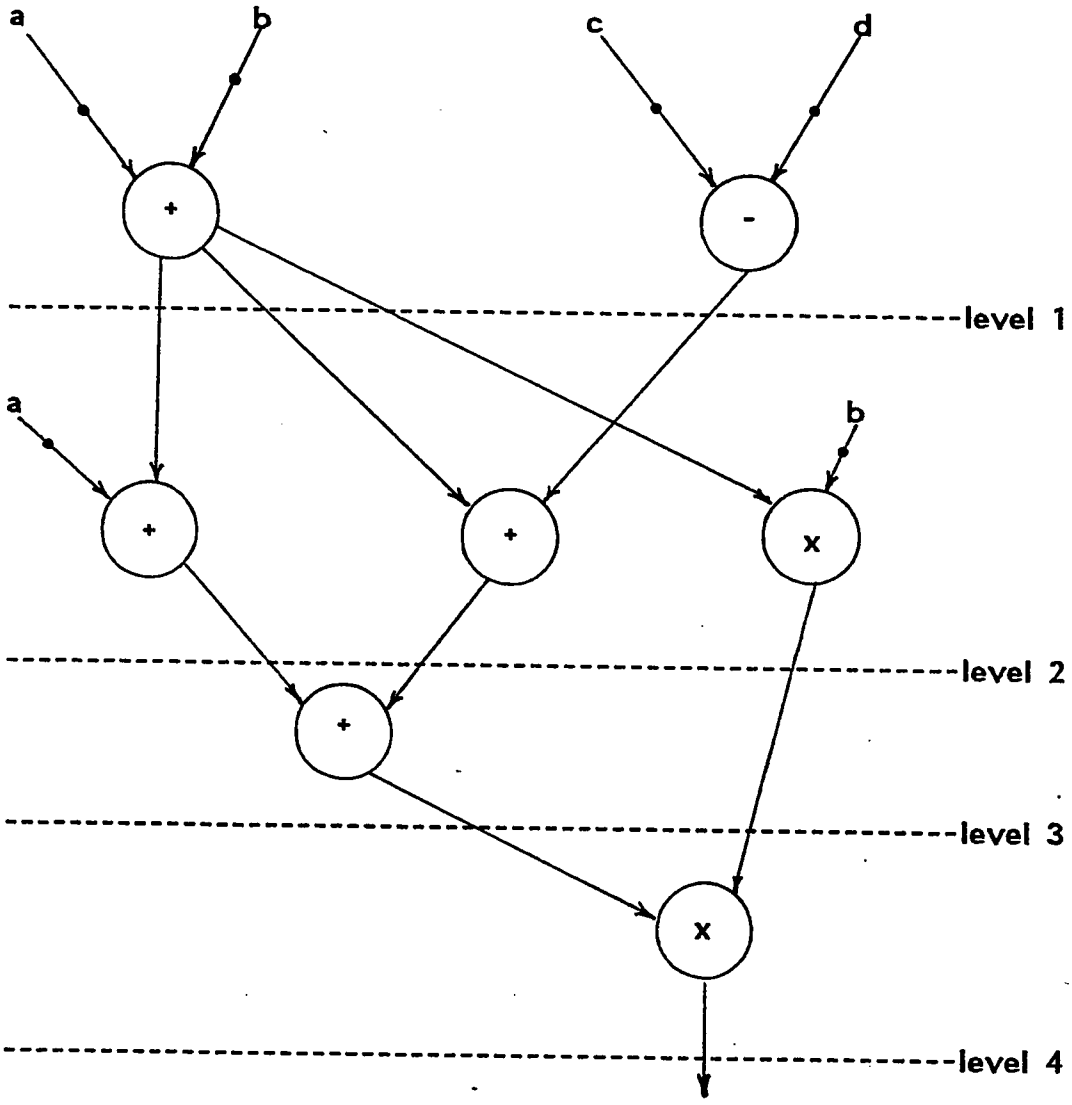


FIGURE 5.7

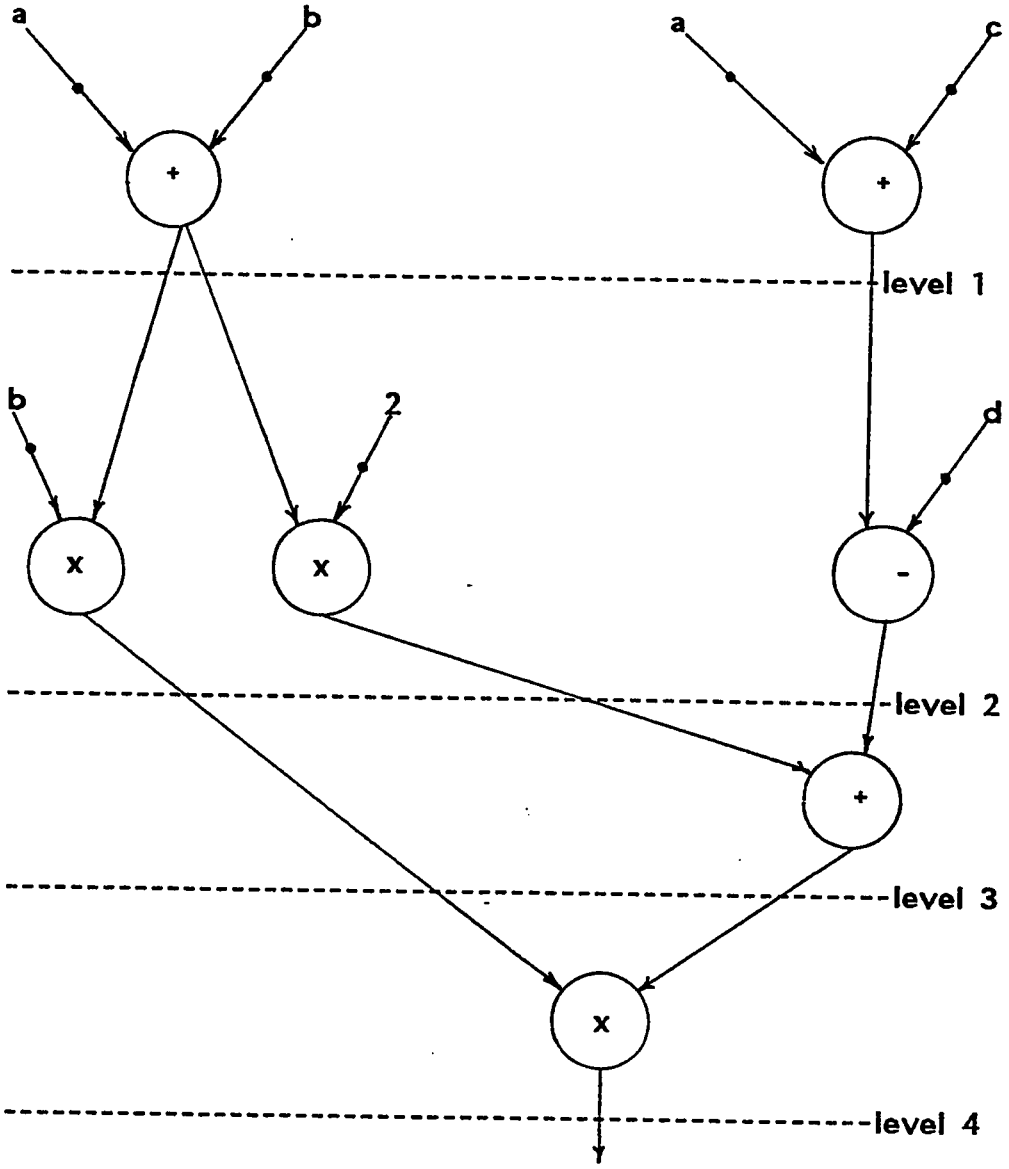


FIGURE 5.8

30 instructions in the form of 30 instructions sequentially one after another. This arrangement will be in the host computer memory, from where the 30 instructions will be loaded into the DFEE system.

5.4 FORMAT OF THE EXPRESSION

The input expression to the optimization compiler will be a standard input format string. This format is described below:

a) Addition, Subtraction, Division, Multiplication are all binary operations. These operations require two operands. Square-root and squaring operations are assumed to be unary operations because they require one operand.

b) The expression will start with an open bracket '(' and end with a closed bracket ')'. Recursively, all operations inside these brackets will have the same format without any blanks in between, e.g.,

(((a+b)x(c-d)/((a+b)x(b/a)))

$(((a)^2+(b)^2)x((a)^2-(c)^2))$

((axb)-(a-c)+(a/c))

c) The open bracket and the closed bracket will not enclose an operand without specified operation; e.g.,

((a)+(b)) is WRONG

(a-b) is CORRECT

5.5 OPTIMIZATION COMPILER: ALGORITHMIC DESCRIPTION

The algorithm description gives the overall design of the optimization compiler. Since the optimization compiler is software implemented and therefore machine-dependent, the description is also at a general level; i.e., data-structures are to be chosen at the time of implementation by the user.

To transfer the 36-bit instructions into the DFEE hardware from the host computer hardware, an interface is required which would link the two hardware systems. This machine level interface can be in the machine-level language of the host computer.

5.5.1 OPTIMIZATION COMPILER ALGORITHM

OPTIMIZATION COMPILER(INPUT:String type A or type B; OUTPUT: Final result loaded into the output variable in the host computer memory.);

(* I N I T I L I Z A T I O N *)

1. Scan for the input variables and output variable in the input expression.Note that the output variable is only one.
2. Retrieve the data values of the input variables from the host computer memory and replace the input variables with these data-values.
3. Store the address of the output variable defined in the input string in working memory space.
4. Scan for LOOP or EXP string.

5. IF LOOP string is found in step 4

THEN

generate the new string of expression with data-values instead of input variables with replacing the y (initial) value into the expression

ELSE

generate the new string of expression with data-values instead of input variables for straight expression

6. IF LOOP string is found

THEN

BEGIN

a. CALL GEN_DATAFLOW_LOOP

(INPUT: String expression input from step 5;

OUTPUT: Data-Flow graph of string expression including the LOOP operation);

b. CALL GEN_DATAFLOW_COUNT

(INPUT : initial value, final value from input of type B;

OUTPUT: Data-Flow graph of loop schema);

c. CALL SYNC_BOOLEANTOKEN

(INPUT : Data-Flow graph from step a and b;

OUTPUT: Modified data-flow graph of loop count with SYNCHRONIZATION instructions);

d. CALL GENLOOP_INSTRUCTION

(INPUT: Data-Flow graph from step a and c;

OUTPUT : Coded instructions);

e. CALL CONSTRAINT CHECKER

```
    (INPUT : Coded instructions;
     OUTPUT: Error signal );
f. IF Error Signal False
    THEN
     BEGIN
      a. CALL INTERFACE
         (INPUT : Coded instructions;
          OUTPUT: Output, DFEE error);
         IF DFEE error true
         THEN
          GO TO ERROR STATE AND ABANDON OPERATION;
         ELSE
          a. Put the output in memory location given by
             output variable address saved at initialization;
          b. QUIT.
    END
    ELSE
     BEGIN
      a. CALL CONSTRAINT_SATISFIER
         (INPUT: Coded instructions;
          OUTPUT: Modified instructions );
      b. Go to step e.
    END;
END
ELSE (*if straight expression*)
 BEGIN
  a. CALL GEN_DATAFLOW
```

```
    ( INPUT : String expression from step 5;
      OUTPUT: Data-Flow graph of expression );
 b. CALL GEN_INSTRUCTION
    ( INPUT : Data-Flow graph from step a;
      OUTPUT: Coded instructions );
 b1. CALL CONSTRAINT CHECKER
     ( INPUT: Coded instructions from step b;
       OUTPUT: Error Signal );
 IF  Error Signal False THEN
    BEGIN
     a. CALL INTERFACE
    ( INPUT: Coded instructions;
      OUTPUT: Output , DFEE error);
  IF DFEE error true THEN
   GO TO ERROR STATE AND ABANDON OPERATION.
  ELSE
   a. Put the output in the memory location given
      by the output variable address saved at initilization.
   b. QUIT.
END;
ELSE
 BEGIN
  a. CALL CONSTRAINT_SATISFIER
     ( INPUT : Coded instructions;
       OUTPUT: Modified instructions );
  b. GO TO step b1.
 END;
```

END;

7. QUIT.

5.5.2 DEFINITION OF PROCEDURE CALLS

GEN_DATAFLOW_LOOP

This routine takes the expression as input and generates the data-flow graph for the loop schema. This data-flow graph is a computation data-flow graph; i.e., it represents the recursive computation. Included in this data-flow graph is the loop operation, since the result needs to be conditionally transfered to two instructions depending on the boolean token. This routine will also check for the bracket mismatch, if it exists in the input expression.

GEN_DATAFLOW_COUNT

This routine inputs the initial value and the final value of the loop iteration, i.e., for I:=initial value to final value. The routine generates the corresponding data-flow graph for counting the iterations. This is the data-flow graph, from where the boolean token will emerge as the result of the COMP(are) instruction. Depending on this boolean token, either the count would be added or when the final value is reached, it would be left as it is. This boolean token will be communicated (by address) to the data-flow graph generated by the GEN_DATAFLOW_LOOP.

SYNC_BOOLEANTOKEN

This routine will generate the synchronization tokens which are needed to synchronize the levels of computation data-flow graph and the iteration count data-flow graph (Figure 2.9).

GEN_LOOP_INSTRUCTION

From the computation data-flow graph and the iteration data-flow graph, this routine will generate the coded instructions for all the 30 instruction cells. The routine must satisfy the following requirements:

a) Blank instructions for completion of memory cell blocks.

b) Blank operand insertion into instructions.

c) Blank instruction placement for addressing more than two instructions.

d) FIN(ish) opcode instruction for termination of execution.

e) Optimal generation of instructions.

This routine will also generate the instructions for LOOP and COMP instructions. These opcodes are exclusively for LOOP schema.

CONSTRAINT CHECKER

This routine will scan all the 30 instruction cells in pairs of two; one from Cell Block I and one from Cell Block II. It will detect the two constraints mentioned. If these constraints are not satisfied, it will give an error message to the other routine.

INTERFACE

The interface will be a host computer machine-level software routine. It will transfer everyone of the 30 instructions into a 36 bit vector according to the coded instructions. It will organize the memory of the host computer, such that these instructions form a 30 instruction stack to be down-loaded into the DFEE memory cell blocks.

Once the loading is done, the routine will free the host computer CPU for multi-tasking while enabling the interrupt mechanism for the CPU. As soon as the output or the computation error message is received from the DFEE as an interrupt, the CPU will return to the interface routine for further action.

CONSTRAINT SATISFIER

This routine will come into play when the constraints are not satisfied by the instruction placements in the cell blocks. This routine will detect the placement of a faulty instruction and correct it by switching the instruction cell contents. Once the instruction placement is modified, it will change all the addresses according to the new placements.

GEN_DATAFLOW

For the case of straight expressions, this routine will examine and detect the bracket mismatch for the input expression. It will generate the data-flow graph representing the expression. This routine is

implemented in Appendix D. The data-flow graph is represented in the form of a matrix, i.e., a linear data-structure. Dynamic memory structure (pointers) is used for handling the string expression input.

GEN_INSTRUCTION

From the computation data-flow graph this routine will generate the coded instructions for all the 30 instruction cells inside the DFEE architecture. This routine will take care of :

1) Blank instructions inserted to fill the memory cell blocks.
2) Blank operands to be inserted in the instructions.
3) Blank instructions to be inserted for addressing more than two instructions.
4) FIN opcode for termination of data-flow computation.
5) Optimal generation of instructions.

The implementation of this routine is shown in Appendix E. However, the implementation only takes into account the optimality (5) reqirement.

CHAPTER 6

APPLICATION AND CONCLUSION

This chapter deals with applications of the DFEE chip. Drawbacks with the DFEE system are presented in this chapter. Lastly, conclusions and future research issues are presented.

6.1 APPLICATION OF THE DFEE CHIP

The applications of the DFEE system can be classified into two categories:

i) Signal processsing.

ii) Iterative algorithms which require expression evaluation.

iii) DFEE chip as a building block processor for a loosely coupled array processor system, i.e., the DFEE chips do not require inter-processor communication or data-sharing.

i) Signal processing application

Consider the 2nd order difference equation relating the input and output of a 2nd order recursive digital filter:

$$Y(n) := a1 \times y(n-1) + a2 \times y(n-2) + b0 \times X(n) \qquad (1)$$

The above equation can be interpreted as a computational algorithm in which the delayed values of the output are multiplied by the coefficients a1, a2 while the input is multiplied by the coefficient

b0 without delay. The addition and multiplication operations are closed in the integer field. Therefore, the spectrum of output samples will also be in the integer field.

A typical user program for computing (1) may look like this:

```
WHILE stop false DO
BEGIN
n:=0;
B:=y(n-1);
C:=y(n-2);
D:=X(n);
E:=b0;
F:=a1;
G:=a2;
OPTCOMPILER [ INPUT:  exp(((FxB)+(GxC))+(ExD));
OUTPUT: A];
y(n):=A;
n:=n+1;
END.
```

The input samples and the initial output samples are assumed to be inside the host computer memory. An array structure is shown in the program to read them. The stop represents the completion of the input samples.

ii) Iterative algorithms which require expression evaluation

The DFEE system supports loop schema. Therefore any computation algorithm which uses the following recursive expression can utilize the DFEE chip:

Y(i+1) := f(set of constant input literals, Y(i)) (2)

for i:=initial value to final value (2a)

where f is any function in terms of ordinary arithmetic operations.

This can be used in numerical number crunching applications where the host computer has to go through several iterations till a value is below a specified error.

Consider the following program

```
DO UNTIL error=min
BEGIN
OPTCOMPILER ( INPUT : loop [ ( (a-b) x (a+b))+(y-d)];
y(initial); initial value; final value; OUTPUT: y);
error:=y;
Y(initial):=y;
END.
```

In the above program, after each iteration done by the DFEE chip, the host computer program compares the output 'y' with min. If these values are not equal then the output 'y' is put back into the initial value of y, y(initial), and sent to the DFEE chip with the same loop iteration, initial value and the final value.

iii) DFEE chip as a building block for loosely coupled array processor system

The DFEE chip can be viewed as a machine which can execute data-flow graph represented instructions. Since the DFEE chip can execute the major arithmetic operations in integer fixed point arithmetic, it can also be viewed as an advanced arithmetic logic unit. The DFEE chip does not need any kind of memory co-operation from its master, the host computer. The fact that it can execute data-flow graph represented computation makes it extremly time efficient since it involes inherent parallelism in its execution control [3]. In addition, the execution is free from side-effects, i.e., there is no shared memory in the DFEE chip.

By extending the architecture of the DFEE-host computer, an array processing system can be envisioned. The host computer has several DFEE chips under its control. Each DFEE chip is independent of the other chip. Each DFEE chip views the host computer as its communicator and controller.

By definition, an array processor is a synchronous parallel computer with multiple arithmetic logic units, called processing elements (PE). These PE can operate in parallel in a lockstep manner under a control unit [1]. The DFEE chips around the host computer can be viewed as the parallel processors and the host computer can be viewed as the control unit.

Figure 6.1 shows a conventional array processor architecture. Figure 6.2 shows the array processor system built around the

DFEE chip as a building block. Figure 6.2 exhibits a major difference from Figure 6.1. There is no PE interconnection network between the processors, i.e., the DFEE chips. The only communication is a two-way link between the host computer (the control unit) and the PE (the DFEE chips).

The array processing system is suitable for vector processing, i.e., the same set of operations are done on a different set of data which constitute a vector. The host computer can generate the same instructions with the same set of operations , but insert different operands in them. Each of these instructions can be destined to the DFEE chips.

Since one DFEE chip would be loaded at one time, a parallel START signal can be issued from the host computer to synchronously start the DFEE processors, once all the DFEE processors have been loaded.

At the time of output, all the DFEE chips would finish synchronously due to the same number of operations. The host computer (the controller) can load this vector output into its memory for further use.

This is two fold parallelism, one at the operational level of the DFEE chip, the other at the architectural level of the host-computer and DFEE array processing system.

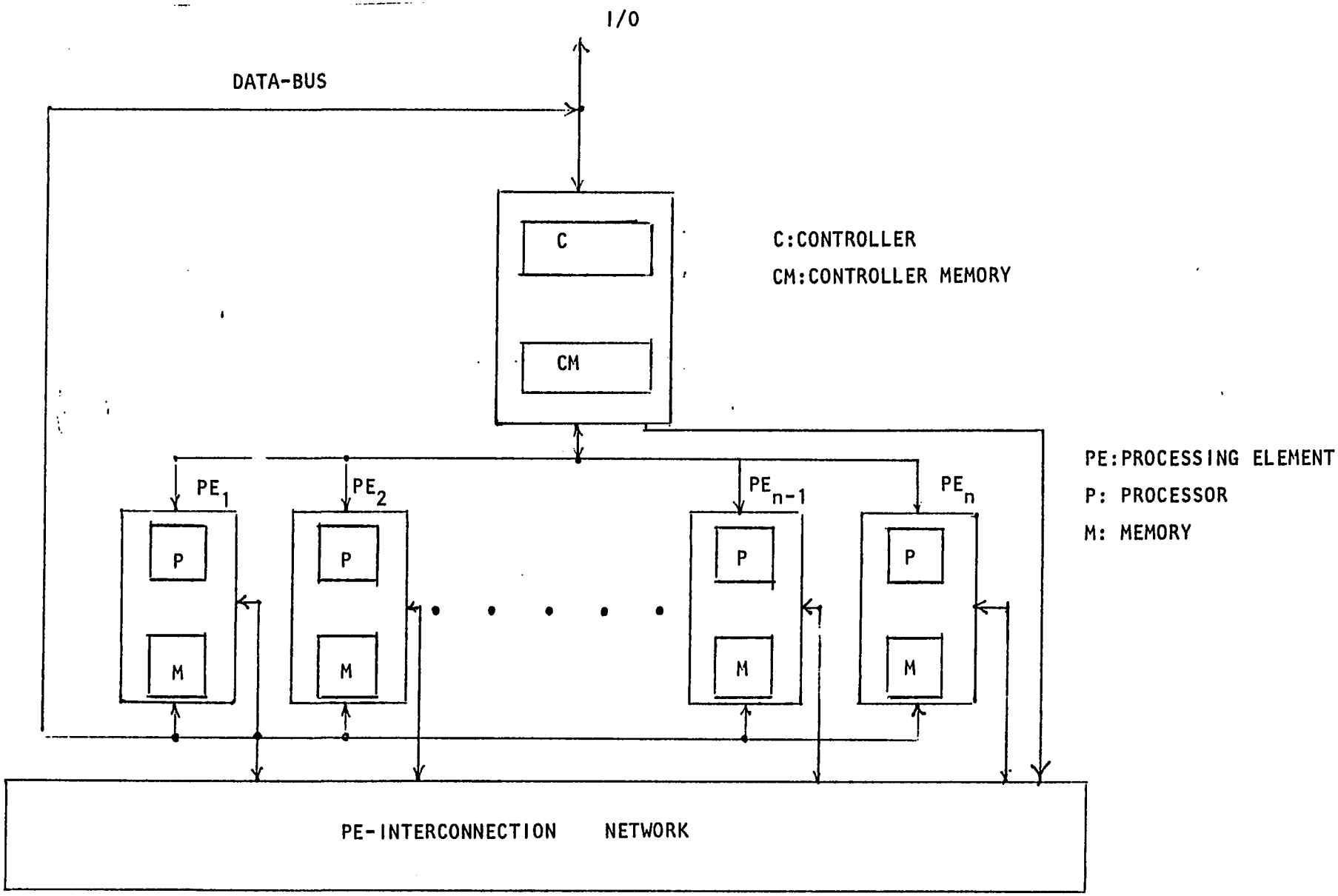


FIGURE 6.1:CONVENTIONAL ARRAY PROCESSOR SYSTEM

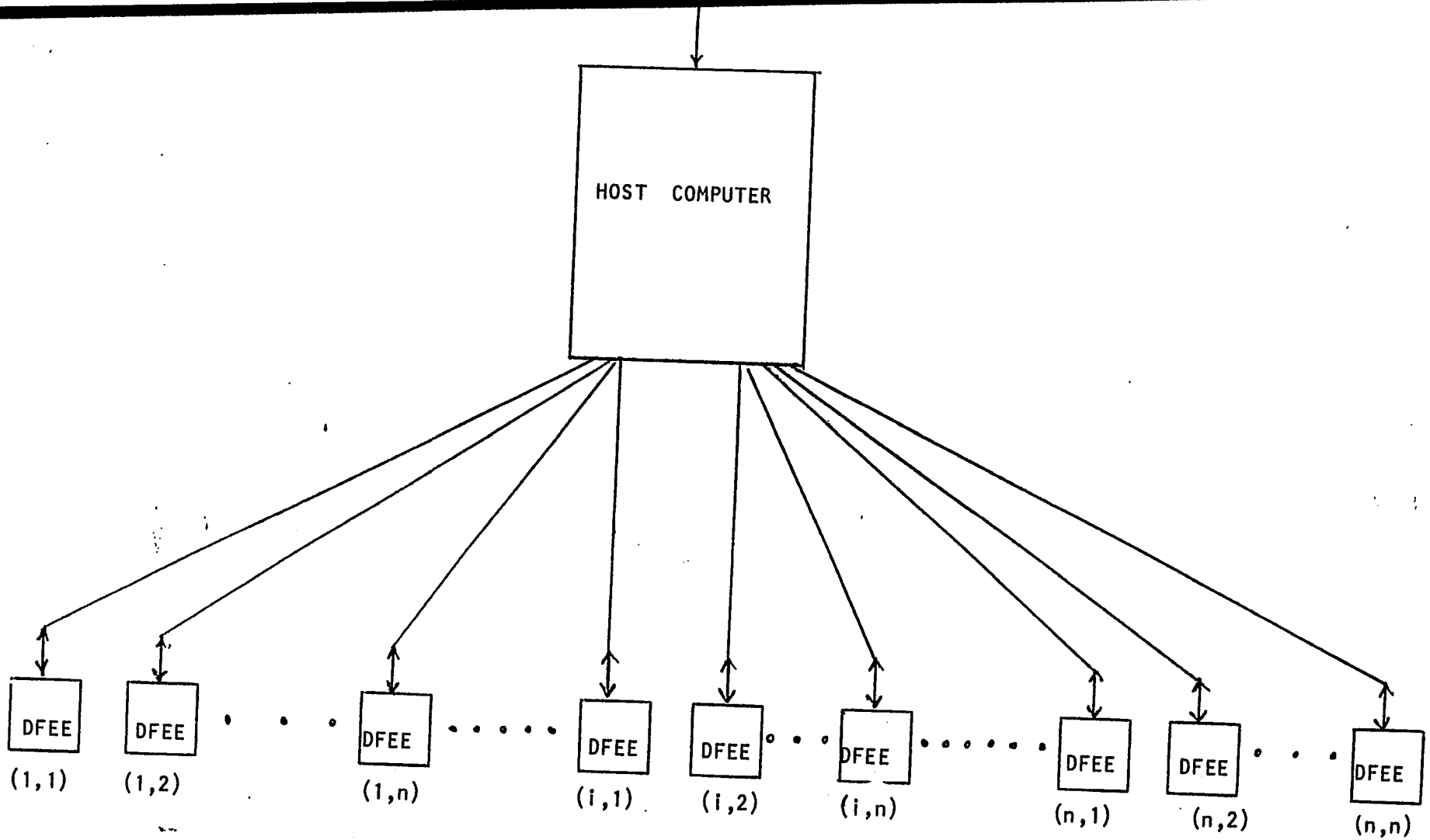


FIGURE 6.2:DFEE ARRAY PROCESSOR SYSTEM

As an illustrative example, consider the multiplication of two nxn matrices. Let ($A=a_{i,k}$) and ($B=b_{k,j}$) be a nx1 and 1xn matrix respectively. The multiplication of A and B generates a product matrix

C=AxB

where $C=c_{i,j}$ is of dimension nxn.

The elements of the product matrix C are related to the elements of A and B by:

$$c_{i,j}=\sum_{k=1}^{n} a_{i,k} x b_{k,j}$$

for i,j=1 to n

Assume that we have nxn DFEE chips in our architecture. To each of these chips the host computer sends the instructions generated from the data-flow graph of the expression

$$\sum_{k=1}^{n} a_{i,k} x b_{k,j}.$$

The data-flow graph for this expression is shown in Figure 6.3. Since the input variables a and b are elements of an nxn array, their values are obtained by the host computer from its memory prior to inputting the instructions into the DFEE chips.

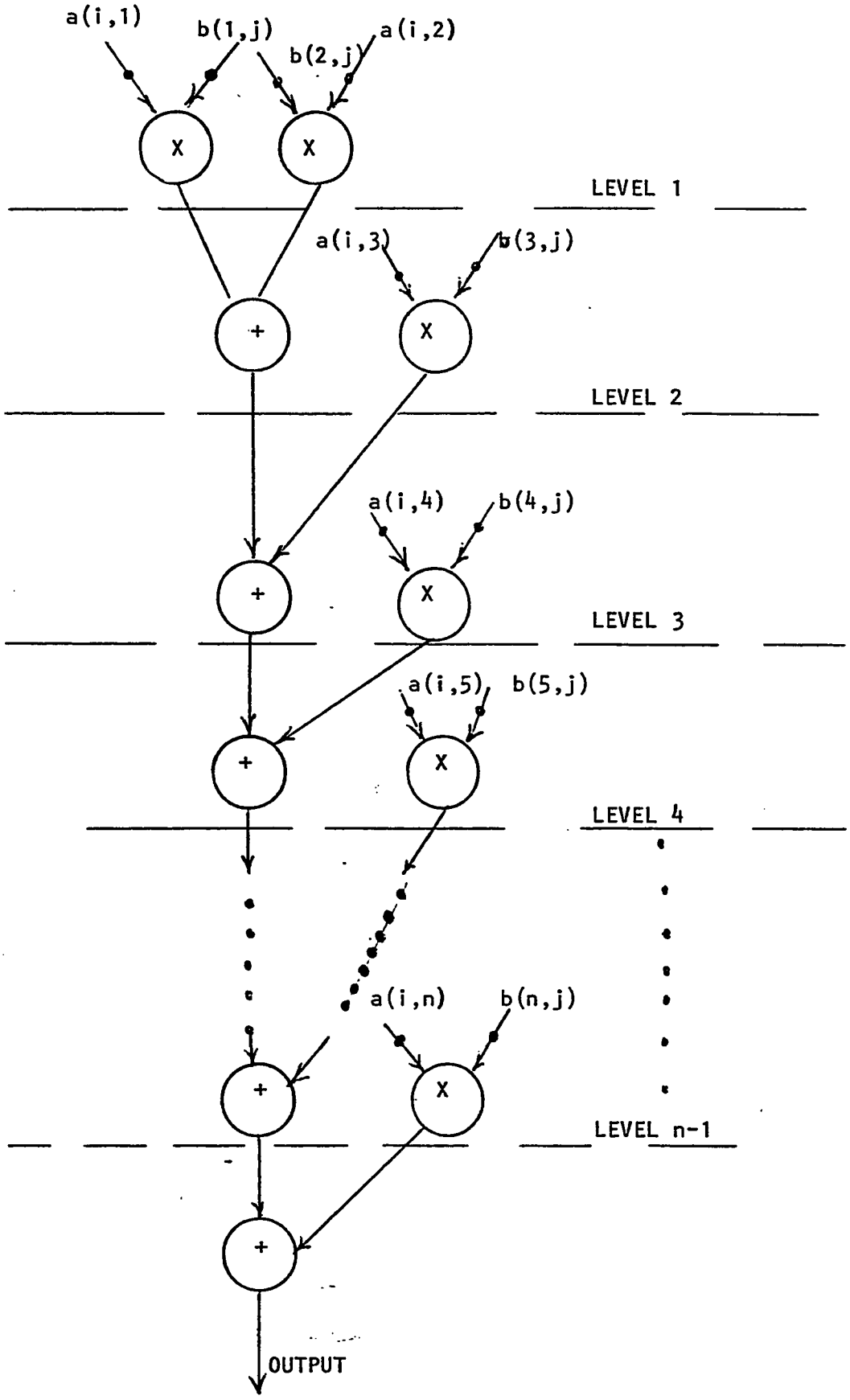


FIGURE 6.3:DATA-FLOW GRAPH FOR MATRIX MULTIPLICATION -214-

Each set of instructions are loaded into one of the nxn DFEE chips (i,j) as shown with index in Figure 6.3. In this way, a two-fold parallelism is achieved in the array processor system.

6.2 DRAWBACKS WITH THE DFEE SYSTEM

The DFEE system requires efficient software support. If we view the DFEE system as another independent hardware, then its dependence on the software support is justified. However, since the DFEE is an add-on special purpose chip , which enhances the performance of the host computer, its dependence on the software support is an overhead.

The time gained in using the parallelism in the DFEE system can be lost in prior arrangement of instructions and going over them repeatedly to make sure that the constraints are satisfied. However, the constraints imposed by the architecture will exist and should be considered as a part of the trade-off. The constraints can be relaxed if we increase the IC Address Queue and Valid Address Queue length. But this results in severe degradation of time performance due to queue delays.

The solution to this may lie in designing an additional processor which would do the function of the optimization compiler. Whether an additional processor can be safely accomodated on the same DFEE chip can be considered after the implementation phase of the DFEE chip.

The other drawback which exists in the DFEE system is the absence of an overall pipelined mechanism; that is, there cannot be a continuous flow of expression inputs along with the input variables and continuous outputs coming out of the DFEE system. The reason for this is, once the DFEE system is in the executing phase, it forms an autonomous computing system. In this phase, no communication in the form of an input or output is possible from it. If this execution mechanism is interrupted, it would no longer be data-flow environment. The pipelining and parallelism is utilized at the fine-grain granuleity of computation expression rather then at course-grain granuleity of the computation at the system level [3].

6.3 FUTURE RESEARCH ISSUES AND GUIDELINES TO PROBE FURTHER

Ever since the data-flow concept was conceived in the early 70's, much interest has been shown in this field. It has been stated that data-flow computing architectures would be the leading candidate for the fifth generation computers [5]. Towards the practical realization of a general-purpose data-flow computer, below are a number of technical problems that need to be solved [1]:

1) The design of a high-level data-flow language.
2) The design of communication networks between the processing elements of the data-flow computer.
3) The development of processing elements and structured

memories.

4) The development of overall control mechanisms and operating system for data-flow computer.
5) Overhead estimations of data-flow computers, including development overhead, execution overhead and application overhead.
6) Performance analysis of data-flow computers for irregular parallelism with intermix of scalar and vector computations.

The above points are general guidelines. However, towards the continuation of this thesis work, following are the future research considerations:

1) Development of an array processing system based around a control unit (the host computer) with identical DFEE chips as the the processing elements.
2) Increasing the operational capabilities of the DFEE processing element to include logical and string operations.
3) Inclusion of data-structures like arrays and strings in the DFEE system.
4) Development of an efficient instruction generator with optimality considerations, i.e., optimization compiler.
5) Study of computation expression forms most widely used, so that the DFEE architecture restrictions can be relaxed.
6) Extension of the DFEE architecture to more than two processor and cell block memory modules.
7) Performance analysis of the DFEE system with time.
8) VLSI implementation of the DFEE chip with complete layout

and description in a layout language ready for fabrication.

9) Inclusion of fault tolerance in the DFEE architecture.

6.4 CONCLUSION

The purpose of this work was to design a chip which computes computation intensive expressions in a data-flow environment. It is shown in this work that the architecture which does this is highly suitable for VLSI implementation. It exhibits regular data-paths, modularity, and regularity in structures of VLSI.

This work explores the design of a highly pipelined universal processing element specially suited for data-flow environment. Therefore, this work covers one of the major design issues towards data-flow computing systems.

The thesis work also states the trade-offs in the design phase of such an architecture. Accordingly, a software support is defined in the form of an optimization compiler which takes into consideration the constraints imposed by the DFEE architecture.

Lastly, this work shows the vitality of such data-flow architecture for massively parallel computing structures when the DFEE chip is used as a building block processing element.

In conclusion, data-flow offers a viable approach to improve today's computer performance. The development of data-flow computers is in its infancy stage. With the push of VLSI

computing structures and its relation to data-flow environment as pointed out in this work, we can anticipate an important role of data-flow mechanisms and their variations in future computers. This work has, therefore, been an illustration of such novel computational concept.

REFERENCE

1) Hwang, K.: COMPUTER ARCHITECTURE AND PARALLEL PROCESSING. McGraw-Hill, New York, 1984.

2) IEEE COMPUTER: Special issue on Parallel Processing. June 1985.

3) Haritimo, I.: DFSP-A Data Flow Signal Processor. IEEE TRAN. ON COMPUTERS, pp 23-33, January 1986.

4) Gaudiot, J. L.: Structure Handling in Data-Flow Computers. IEEE TRAN. ON COMPUTERS, C-35, pp 489-502, June 1986.

5) Dennis,J. B.: Supercomputers. IEEE COMPUTER, pp 48-57, January 1987.

6) Haritimo, I.: On the use of Data-Flow Principles in Digital Signal Processing. IEEE CONFERENCE ON INTEGRATED CIRCUITS, pp 453-458, 1981.

7) Tiberghien, J.: NEW COMPUTER ARCHITECTURE, Academic Press, London, 1984.

8) Weitz, E and J. Katzenelson: VLSI Simulation and Data Abstraction. IEEE TRAN. ON COMPUTER AIDED DESIGN OF INTEGRATED CIRCUITS. CAD-5, pp 371-378, July 1986.

9) Treleaven, P. C.: VLSI Architecture. IEEE COMPUTER, pp 33-45, June 1982.

10) Patterson, D. A. and C. H. Sequin: Design Considerations for Single Chip Computers. IEEE TRAN. ON COMPUTERS, C-29, pp 108-115, February 1980.

11) Kamal, A. K. and H. Singh: A Generalized Pipeline Array. IEEE TRAN. ON COMPUTERS, C-23, pp 533-535, May 1974.

12) Long, T. and M. D. Ercegovac: DIGITAL SYSTEMS AND HARDWARE FIRMWARE ALGORITHMS. John Wiley and Sons, New York, 1985.

13) Packwell, D. and K. Eshraghian: BASIC VLSI DESIGN. Prentice-Hall of Australia Pty Ltd, 1985.

14) Dennis, J.: A Computer Architecture for Highly Parallel Signal Processing. PROJECT MAC. Massachusetts Institute of Technology.

15) Trelevean, P. C. and D. R. Brownbridge: Data-Driven and Demand-Driven Computer Architecture. ASM COMPUTING SURVEY, Vol 14, NO 1, pp 143-168, 1982.

16) Dias, D. M. and R. Jump: Packet Switching Interconnection Networks for Modular Systems. IEEE COMPUTER, Vol 14, pp 43-54, 1981.

17) Oppenheim, A. V. and R. Schaffer: DIGITAL SIGNAL PROCESSING. Prentice-Hall, Englewood, New Jersey, 1975.

18) Muroga, S.: VLSI SYSTEM DESIGN. John Wiley and Sons, New York , 1982.

19) Hwang, K.: COMPUTER ARITHMETIC. Prentice-Hall, Englewood, New Jersey, 1979.

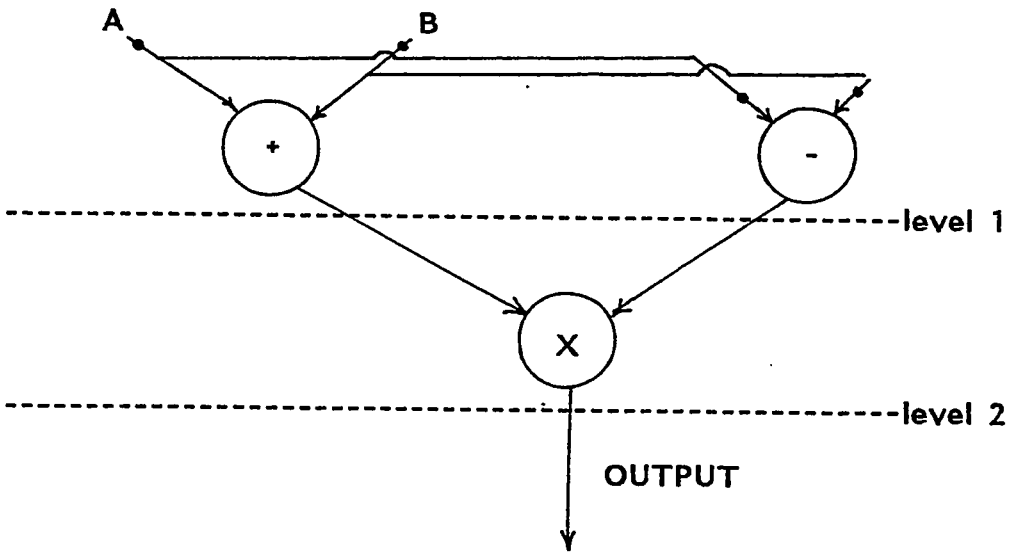
20) Mead, C. and L. Conaway: INTRODUCTION TO VLSI SYSTEMS. Addison-Weslely Publishing Company, New York, 1980.

21) Mano, M. M: COMPUTER SYSTEM ARCHITECTURE. Prentice-Hall, Englewood, New Jersey, 1982.

APPENDIX A

SIMULATION PROGRAM FOR Straight expression

COMPUTATION: OUTPUT:=(A+B) x (A-B)
A=5, B=4.



```
  1 |  %$P 50000%
  2 |  %$X 50000%
  3 |  PROGRAM SIMARC(OUTPUT);
  4 |  (* COMMENTS REGARDING THE SIMULATION                                    *)
  5 |  (* 1A)THIS SIMULATION IS FOR SIGLE OUTPUT'FIN' EXPRESSION.*)
  6 |  (*     MODIFICATION REQUIRED FOR MULTIPLE OUTPUT INSTRUCTION*)
  7 |  (*     USING OUTPTREDY SIGNALARRY.
  8 |  (* 1) ALL THE VARIABLES ARE GLOBAL.I.E THERE NAME IS COMMUNICATED*)
  9 |  (*     DIRECTLY  TO ALL THE PROCEDURES REPRESENTING THE FUNCTIONALITY*)
 10 |  (*     OF THE SUB-UNITS.E.G FETCH-UNIT,UPDATE-UNIT,PRECES-ELEMENT,*)
 11 |  (*     DISTRIBUTION NETWORK.*)
 12 |  (* 2) THE PROCESSING ELEMENT IS REPRESENTED BY AN ARRAY OF*)
 13 |  (*     PIPELINE LENGTH 4, ALONG WITH THE ADDRESS PIPELINE          *)
 14 |  (*     OF THE SAME LENGTH. FURTHER, THE PR.EL DOES NOT TAKE CARE    *)
 15 |  (*     CARE OF SIGN IN THE SIMULATION.                              *)
 16 |  (* 3) TO REPRESENT THE CLOCK , A  OUTER LOOP IS USED*)
 17 |  (*     WHICH WILL BE EXITED WHEN THERE IS FIN OPCODE FOUND  *)
 18 |  (* 4) IMPORTANT: TO REPRESENT  THE PARALLEL EXECUTION OF CELL BLOCKS*)
 19 |  (*     ANOTHER DO LOOP RUNS THE SAME SET OF PROCEDURES NUMBER OF TIMES*)
 20 |  (*     WHICH ARE EQUAL TO THE CELL BLOCKS.THIS RESIDES INSIDE THE CLOCK*)
 21 |  (*     WHILE LOOP STATEMENT.*)
 22 |  (* 5) THERE IS A PROBLEM ENCOUNTERED WHILE RUNNING THE SIMULATION*)
 23 |  (*     OF PARALLEL EXECUTION OF CB. THE PROBLEM IS THAT CB(ITH) IS *)
 24 |  (*     UPDATING THE ITH +1 IN THE SAME CLOCK. THIS SHOULD BE DONE *)
 25 |  (*     IN THE NEXT CLOCK PHASE. FOR THIS PURPOSE, SIMSAVE AND SIMSAVB*)
 26 |  (*     HAVE BEEN DEFINED SO AS TO MAKE THE PARALLEL EXECUTION SIMULATION*)
 27 |  (* 6) NOTICE THE CONSTRAINT ON OPTIMIZATION COMPILER:*)
 28 |  (*     C O N S T R A I N T  ON THE OPTIMIZATION COMPILER*)
 29 |  (*     INSTRUCTION CELL INSTRUCTIONS ARE PLACED SUCH THAT THE S*)
 30 |  (*     UPADDGIV PIPELINE MAY NOT BE "OVERLOADED" AND LOOSE THE *)
 31 |  (*     ADDRESS OF THE IC JUST UPDATED.IE IF UPADDGIV PIPELINE FOR*)
 32 |  (*     CB(I)OS ALREADY FILLED ,THEN CB(J) WHERE J IS NOT EQUAT I*)
 33 |  (*     CANNOT PUT ANY ADDRESS IN CB(I) PIPELINE OF UPADDVIG WITHOUT*)
 34 |  (*     LOOSING ONE ADDRESS VALUE*)
 35 |  (*************************************************************************)
 36 |  CONST
 37 |   MAXIC=4;(*MAXIMUM NUMBER OF IC IN CELL BLOCK*)
 38 |   MAXCELBLK=2;(*MAX. NUMBER OF CELL BLOCKS IN ARCHITECTURE*)
 39 |   PIPELENGTH=2;(*PROCESSING ELEM. PIPELENGTH*)
 40 |  TYPE
 41 |   SUB1=1..MAXCELBLK;
 42 |   SUB2=1..MAXIC;
 43 |   SUB3=1..PIPELENGTH;
 44 |   NUMCELBLK=0..MAXCELBLK;(*0 JUST TO INCORPORATE NON VALID ADDRESS*)
 45 |                          (*FOR UPDATE UNIT                          *)
 46 |   NUMIC=0..MAXIC;(*ABOVE COMMENT APPLIES*)
 47 |   NUMOPR=0..2;(*SINCE THE NO OF OPR ARE ALWAYS EQUAL TO 2*)
 48 |   CODE=(NOP,ADD,SUB,MUL,DIVI,SQURT,SQAR,COMP,LOOPI,FIN);
 49 |   (*****COMMENT*********************************************)
 50 |   (*NOP SHOULD BE 0000 IN THE DESIGN BUT NOP IS NOT USED IN THE *)
 51 |   (*DESIGN BECAUSE EVERY IC HAS OPCODE OTHER THAN NOP WHEN FILLED*)
 52 |   (*BY OPTCOMPILER                                              *)
 53 |   (*************************************************************)
 54 |   STARVAL='A'..'D';
 55 |   (*************************************************************)
 56 |   (*****2-BIT REPRESENTATION,A=00,B=01,C=10,D=11*****************)
 57 |   (*****USED BY THE UPDATE UNIT*********************************)
 58 |   ADDSPEC=
 59 |       RECORD
 60 |          CB:NUMCELBLK;
```

```
63  |        END;
64  |    INSTRCELL=
65  |        RECORD
66  |           OPCODE:CODE;
67  |           OPRAND1,OPRAND2,RESULT:INTEGER;
68  |           STAR:STARVAL;
69  |           ADDRSLT1,ADDRSLT2:ADDSPEC
70  |        END;
71  |   (*****************************************************************)
72  |   (****EACH CELL BLOCK HAS FOUR INSTRUCTION CELLS. THIS WILL BE REPRESENTED*)
73  |   (****2-D CELL BLOCK ARRAY.SAMILARLY, NUMBER OF PE ARE EQUAL TO CB*)
74  |   (****AND EACH PE HAS 16 PIPELINE LENGTH**************************)
75  |   (****EACH CELL BLOCK FETCH UNIT HAS A COUNTER SO ARRAY************)
76  |   (*****************************************************************)
77  |   VAR
78  |      CELLBLOCK:ARRAY(.SUB1,SUB2.)OF INSTRCELL;
79  |      PROCCALC:ARRAY(.SUB1,SUB3.)OF INSTRCELL;
80  |   (*THIS ARRAY TAKES CARE OF CALCULATED RESULT*****************)
81  |   (**IN THE INITILIZATION OF THE MAIN PROGRAM, THIS ARRAY HAS TO *)
82  |   (**BE SET TO 0 IN ORDER TO SIMULATE THE 16CLOCK PIPLINE*********)
83  |      PROCADDRESS:ARRAY(.SUB1,SUB3.)OF INSTRCELL;
84  |   (**IN THE INITILIZATION OF THE MAIN PROGRAM, THIS ARRAY HAS TO *)
85  |   (**BE SET TO 0 IN ORDER TO SIMULATE THE 16CLOCK PIPLINE*********)
86  |   (**THIS ARRAY TAKES CARE OF SYNCHRONIZING THE RESULT WITH ITS****)
87  |   (***ADDRESS WHICH GOES PAST UNAFFECTED***************************)
88  |      COUNTICNUM:ARRAY(.SUB1.)OF INTEGER;
89  |   (*****************************************************************)
90  |   (*****************************************************************)
91  |   (************DEFINATION OF GLOBAL VARIABLES************************)
92  |   (***WHICH ARE CALLED BY THE SAME NAME IN THE PROCEDURE CALLS********)
93  |   (**********WHY?SEE COMMENT IN THE BEGINING************************)
94  |     II,JJ,STOP,I,J,K,L,RESULTOUT,CBLOOPVAL,CLOCK:INTEGER;
95  |   (*****RESULT OUT IS THE VALUE OF RESULT GIVEN TO THE MASTERCONTROLLER*)
96  |   (*****I,J,K,L ARE INDEX VARIABLES USED AT VARIOUS PLACES**************)
97  |   (***CBLOOPVAL IS USED FOR DETERMININMG THE PARALLEL EXECUTION*****)
98  |   (***I,E CBLOOPVAL:=1 TO MAXCELBLK*********************************)
99  |     OTPTREDY,UPDATYES:ARRAY(.SUB1,SUB1.)OF BOOLEAN;
100 |     FLAGOUT:BOOLEAN;
101 |    (***FLAGOUT WILL BE MADE TRUE WHEN OPCODE "FIN" IS FOUND***)
102 |    (*****************************************************************)
103 |    (************NOTE:WE NEED AN AN ARRAY TO DISTUINGUISH BETWEEN ****)
104 |    (********THE TWO UPDATE YES SIGNALS AND OUTPUT READY SINCE WE ARE**)
105 |    (********USING LOOP FOR SIMULATING PARALLEL EXECUTION******)
106 |   (*********OTPTREDY  IS THE SIGNAL WHICH IS LOW BEFORE WE ENTER INTO**)
107 |   (****FETCH UNIT EXECUTION AND IT IS PUT HIGH BY FETCH UNIT IF "FIN"*)
108 |   (****IS FOUND BY THE FETCH UNIT,ELSE IT REMAINS LOW***************)
109 |   (*****************************************************************)
110 |   (****UPDATYES IS THE SIGNAL IF UDATE UNIT UPDATES THE IC AND THEN**)
111 |   (****SENDS THE CORROSPONDING ADDRESS TO THE FETCH UNIT************)
112 |   (*****THIS ADDRESS CONTAINS THE CELL BLOCK NO. ,IC NO **)
113 |   (**HOWEVER, NOTE THAT NO OPRAND NUMBER IS GIVEN TO THE FETCHUNIT*)
114 |   (***NO NEED TO GIVE THE OPRAND NO TO THE FETCH UNIT**)
115 |   (******ALTHOUGH THE CEL BLOCK IS PRE DETERMINED BY THE DISTRIBUTION*)
116 |   (*NETWORK BUT SINCE DUE TO PARALLEL EXECUTION USING DO LOOP, WE NEED*)
117 |   (* SPECIFY ALL THE ADDRESS FIELD SO THE PROCEDUCRE CAN BE USED FOR*)
118 |   (*ALL CELL BLOCKS*)
119 |   (***UPADDGIVE-IS THE ADDRESSGIVEN BY UPDATE UNIT TO FETCH UNIT OF*)
120 |   (*THE ABOVE MENTIONED FORMAT*)
121 |   (***IT CONTAINS ONLY IC NUMBER THOUGH CB NUMBER IS CARRIED              *)
122 |   (***THROUGH IN SOFTWARE,THIS IS NOT DONE SO IN ACTUALL HARDWARE**)
123 |     UPADDGIV:ARRAY(.SUB1,SUB1.)OF ADDSPEC;
124 |   (*****************************************************************)
125 |     DISADDGIV:INSTRCELL;
```

```
127   |  (*NETWORK TO THE UPDATE UNIT AS TO WHICH CB THE RESULT IS COMING TO*)
128   |  (*AGAIN ,IN ACTUAL HARDWARE, THE CB NO. WILL BE USED UP BY THE *)
129   |  (*DISTRIBUTION NETWORK BUT SINCE IN SOFTWARE SIM. AN ARRAY IS CALLED*)
130   |  (*THEREFORE THE CB NO. WILL GO ALONG AS AN INDEX TO THE ARRAY CELBLK*)
131   |  (********THE FETCH UNIT FORMING OPERATION PACKET FOR THE PE GIVES**)
132   |  (********FTHADDGIV-TO THE PE SO IT CAN FORM THE RESULT PACKET******)
133   |  (*****SINCE THERE ARE TWO ADDRESSES ASSOCIATED WITH IT SO*******)
134   |  (**FTHADGIV-WE NEED A STAR AS WELL .UPADDGIVE NEED NOT GIVE THE**)
135   |  (**STAR SINCE ONLY ONE ADDRESS COMMUNICATED**)
136   |    FTHADGIV:INSTRCELL;
137   |  (****SO WE WILL SPECIFY THE WHOLE ADDRESS BLOCK AS**************)
138   |  (***FTHADGIV.STAR,FTHADGIV.ADDRSLT1.CB,FTHADGIV.ADDRSLT1.IC,**)
139   |  (****FTHADGIV.ADDRSLT1.OPR, AND SAMILARLY FOR ADDRSLT2.*******)
140   |  (*************************************************************)
141   |  (*****THE RESULT PACKET OUT OF THE PRECESSING ELEMENT CONTAINS**)
142   |  (*************************************************************)
143   |  (*RESULT *STAR*ADDRESULT1*ADDRESULT2**)
144   |  (*************************************************************)
145   |  (****THIS GOES TO THE DISTRIBUTION NETWORK AND THEN TO THE UPDATE*)
146   |  (*****UNIT.AGAIN ONE THING TO NOTE IS UPDATE UNIT MAY UPDATE******)
147   |  (****MORE INSTRUCTIONS DEPENDING ON STAR (**CONFLICT SHOULD BE****)
148   |  (****AVOIDED BY THE OPTCOMPILER.I.E WITH THE UPDATE YES THERE SHOULD*)
149   |  (***BE NO TWO INSTRUCTIONS ENABLED IN THE SAME CELL BLOCK AT THE SAME*)
150   |  (****TIME. BUT THE UPDATE SIGNAL SENDS ONLY ONE ADDRESS TO THE FETCH*)
151   |  (*UNIT DEPENDING ON THE STAR .SEE THE FLOWCHART*)
152   |  (*************************************************************)
153   |    RESLTPAC:INSTRCELL;
154   |  (**RESLTPAC IS THE FORMAT OF THE RESULT PACKET AFTER THE RESULT IS **)
155   |  (*OUT OF THE PROCESSING ELEMENT ANDIS SYN. WITH THE ADDRESS.*********)
156   |    FETCHREG:ARRAY(.SUB1.)OF INTEGER;
157   |  (*THIS REGISTER WILL RESIDE IN EACH FETCH UNIT AND WILL MAKE SURE**)
158   |  (**THAT SAME IC IS NOT FETCHED "MORE THAN ONCE IN A ROW"*********)
159   |    SIMSAVE:ARRAY(.SUB1.)OF INTEGER;
160   |    SIMSAVB:ARRAY(.SUB1.)OF BOOLEAN;
161   |  (**  ONLY USED IN ORDER TO SOLVE THE PROBLEM OF PARALLEL EXECUTION*(
162   |  (**  SO THAT CBI DOES NOT EXECUTE CB(I+1) IN THE SAME CLOCK**)
163   |  (******I M P O R T A N T*************************************)
164   |  (*** NOT TO BE REPRESENTED IN HARDWARE*******)
165   |  (********FUNCTIONAL DEFINATION OF THE SUB-UNITS REPRESENTED BY THE **)
166   |  (**REMEMBER , COUNTICNUM=1,OTPTREDY=FALSE,UPDATYES=FALSE INITIALLY***)
167   |  (***BEFORE THE EXECUTION OF THE PROCEDURE AND AFTER THE EXECUTION OF*)
168   |  (***THE PROCEDURE**THIS WOULD BE DONE IN THE MAIN PROGRAM ***********)
169   |  (***PROCEDURES STARTS HERE*************************************)
170   |  (***REMEMBER:ALL VAR GLOBALI.E DIRECTLY AFFECT THE MAIN PROGRAM**)
171   |  PROCEDURE FETCHUNIT;
172   |   VAR
173   |      IL:INTEGER;
174   |  BEGIN(*111*)
175   |   FOR IL:=1 TO MAXCELBLK DO
176   |    BEGIN
177   |   WRITELN('++++D E B U G+++++');
178   |   WRITELN('FETCH UNIT PIPLINE UP YES',UPDATYES(.CBLOOPVAL,IL.));
179   |   WRITELN('FET UNIT PIPLINE UPADDRESS',UPADDGIV(.CBLOOPVAL,IL.).IC);
180   |   WRITELN('++++D E B U G+++++')
181   |    END;
182   |   BEGIN(**PROCEDUREUPDATE*)
183   |   J:=CBLOOPVAL;(*J REMAINS SAME FOR FETCH UNIT*)
184   |   IF UPDATYES(.CBLOOPVAL,MAXCELBLK.) THEN
185   |     BEGIN(*TRUE UPDATEYES*)
186   |       K:=UPADDGIV(.CBLOOPVAL,MAXCELBLK.).IC;
187   |       IF(CELLBLOCK(.J,K.).OPCODE=FIN) THEN
188   |         BEGIN(*FIN *)
189   |          OTPTREDY(.CBLOOPVAL,MAXCELBLK.):=TRUE;
190   |          FLAGOUT:=OTPTREDY(.CBLOOPVAL,MAXCELBLK.);
```

```
192 |           WRITELN('FIN OPCODE FOUND FOR CELL':18,CBLOOPVAL:3);
193 |           RESULTOUT:=CELLBLOCK(.J,K.).OPRAND1;
194 |           WRITELN('THE RESULT IS':15,RESULTOUT:3);
195 |           WRITELN('************************************************');
196 |           CELLBLOCK(.J,K.).OPCODE:=NOP;
197 |           CELLBLOCK(.J,K.).OPRAND1:=99;
198 |           CELLBLOCK(.J,K.).OPRAND2:=99;
199 |           CELLBLOCK(.J,K.).RESULT:=99;
200 |           CELLBLOCK(.J,K.).STAR:='A';
201 |           CELLBLOCK(.J,K.).ADDRSLT1.CB:=0;
202 |           CELLBLOCK(.J,K.).ADDRSLT1.IC:=0;
203 |           CELLBLOCK(.J,K.).ADDRSLT1.OPR:=0;
204 |           CELLBLOCK(.J,K.).ADDRSLT2.CB:=0;
205 |           CELLBLOCK(.J,K.).ADDRSLT2.IC:=0;
206 |           CELLBLOCK(.J,K.).ADDRSLT2.OPR:=0
207 |         END(*FIN*)
208 |        ELSE
209 |         BEGIN(*NOT EQ TO FIN*)
210 |         IF(((CELLBLOCK(.J,K.).OPCODE)=SQURT)OR ((CELLBLOCK(.J,K.).OPCODE
211 |            =SQAR)))THEN
212 |   (**NOTICE THIS IS ONLY TESTING FOR OPRAND 1 TO BE FULL FOR **)
213 |   (*****UNI-OPERATION CODE***R-FF HAS TO HAVE PROVISION FOR THIS******)
214 |           BEGIN(*AA*)
215 |            IF((CELLBLOCK(.J,K.).OPRAND1 <> 99))THEN
216 |              BEGIN(*A1*)
217 |               IF(K<>FETCHREG(.J.)) THEN
218 |                BEGIN(*A2*)
219 |                 FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
220 |                 FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
221 |                 FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
222 |                 FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
223 |                 FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
224 |                 FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
225 |                 FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
226 |                 FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
227 |                 FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
228 |                 FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
229 |                 FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
230 |                 FETCHREG(.J.):=K
231 |               END(*A2*)
232 |                ELSE
233 |       BEGIN(*A2 FALSE*)
234 |        K:=COUNTICNUM(.J.);
235 |         IF (K<= MAXIC)THEN
236 |          BEGIN (*BB1*)
237 |           COUNTICNUM(.J.):=COUNTICNUM(.J.)+1;
238 |         IF(((CELLBLOCK(.J,K.).OPCODE)=SQURT)OR ((CELLBLOCK(.J,K.).OPCODE
239 |            =SQAR)))THEN
240 |           BEGIN(*BB2*)
241 |            IF((CELLBLOCK(.J,K.).OPRAND1 <> 99))THEN
242 |             BEGIN(*BB3*)
243 |              IF(K<>FETCHREG(.J.))THEN
244 |               BEGIN(*BB4*)
245 |                FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
246 |                FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
247 |                FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
248 |                FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
249 |                FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
250 |                FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
251 |                FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
252 |                FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
253 |                FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
254 |                FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
255 |                FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
```

```
257 |                    END(*BB4*)
258 |                   ELSE
259 |                    BEGIN(*BB4 FLASE*)
260 |                     FTHADGIV.OPCODE:=NOP;
261 |                     FTHADGIV.OPRAND1:=0;
262 |                     FTHADGIV.OPRAND2:=0;
263 |                     FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
264 |                     FTHADGIV.STAR:='A';
265 |                     FTHADGIV.ADDRSLT1.CB:=0;
266 |                     FTHADGIV.ADDRSLT1.IC:=0;
267 |                     FTHADGIV.ADDRSLT1.OPR:=0;
268 |                     FTHADGIV.ADDRSLT2.CB:=0;
269 |                     FTHADGIV.ADDRSLT2.IC:=0;
270 |                     FTHADGIV.ADDRSLT2.OPR:=0
271 |                    END;(*BB4 FALSE*)
272 |                  END(*BB3*)
273 |                 ELSE
274 |                  BEGIN(*BB3 FALSE*)
275 |                     FTHADGIV.OPCODE:=NOP;
276 |                     FTHADGIV.OPRAND1:=0;
277 |                     FTHADGIV.OPRAND2:=0;
278 |                     FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
279 |                     FTHADGIV.STAR:='A';
280 |                     FTHADGIV.ADDRSLT1.CB:=0;
281 |                     FTHADGIV.ADDRSLT1.IC:=0;
282 |                     FTHADGIV.ADDRSLT1.OPR:=0;
283 |                     FTHADGIV.ADDRSLT2.CB:=0;
284 |                     FTHADGIV.ADDRSLT2.IC:=0;
285 |                     FTHADGIV.ADDRSLT2.OPR:=0
286 |                  END;(*BB3 FALSE*)
287 |               END(*BB2*)
288 |              ELSE(*BB2 FALSE*)
289 |               BEGIN(*BB2F1*)
290 |            IF((CELLBLOCK(.J,K.).OPRAND1<>99)AND(CELLBLOCK(.J,K.).OPRAND2<>99)) THEN
291 |               BEGIN(**BB2 F2*)
292 |                IF(K<>FETCHREG(.J.))THEN
293 |                 BEGIN(**BB2 F3*)
294 |                     FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
295 |                     FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
296 |                     FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
297 |                     FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
298 |                     FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
299 |                     FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
300 |                     FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
301 |                     FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
302 |                     FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
303 |                     FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
304 |                     FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
305 |                     FETCHREG(.J.):=K
306 |                  END(*BB2 F3*)
307 |               ELSE(*BB2 F3 FLASE*)
308 |                BEGIN(*BB2 F3 FALSE*)
309 |                     FTHADGIV.OPCODE:=NOP;
310 |                     FTHADGIV.OPRAND1:=0;
311 |                     FTHADGIV.OPRAND2:=0;
312 |                     FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
313 |                     FTHADGIV.STAR:='A';
314 |                     FTHADGIV.ADDRSLT1.CB:=0;
315 |                     FTHADGIV.ADDRSLT1.IC:=0;
316 |                     FTHADGIV.ADDRSLT1.OPR:=0;
317 |                     FTHADGIV.ADDRSLT2.CB:=0;
318 |                     FTHADGIV.ADDRSLT2.IC:=0;
319 |                     FTHADGIV.ADDRSLT2.OPR:=0
320 |                END;(*BB2 F3 FALSE*)
```

```
322 |            ELSE(*BB2 F2 FALSE*)
323 |             BEGIN(*BB2 F2 FALSE*)
324 |                    FTHADGIV.OPCODE:=NOP;
325 |                    FTHADGIV.OPRAND1:=0;
326 |                    FTHADGIV.OPRAND2:=0;
327 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
328 |                    FTHADGIV.STAR:='A';
329 |                    FTHADGIV.ADDRSLT1.CB:=0;
330 |                    FTHADGIV.ADDRSLT1.IC:=0;
331 |                    FTHADGIV.ADDRSLT1.OPR:=0;
332 |                    FTHADGIV.ADDRSLT2.CB:=0;
333 |                    FTHADGIV.ADDRSLT2.IC:=0;
334 |                    FTHADGIV.ADDRSLT2.OPR:=0
335 |             END;(*BB2 F2 FALSE*)
336 |           END;(*BB2 F1*)
337 |          END(*BB1*)
338 |          ELSE(*BB1 FALSE*)
339 |           BEGIN(*BB1 FALSE*)
340 |                    FTHADGIV.OPCODE:=NOP;
341 |                    FTHADGIV.OPRAND1:=0;
342 |                    FTHADGIV.OPRAND2:=0;
343 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
344 |                    FTHADGIV.STAR:='A';
345 |                    FTHADGIV.ADDRSLT1.CB:=0;
346 |                    FTHADGIV.ADDRSLT1.IC:=0;
347 |                    FTHADGIV.ADDRSLT1.OPR:=0;
348 |                    FTHADGIV.ADDRSLT2.CB:=0;
349 |                    FTHADGIV.ADDRSLT2.IC:=0;
350 |                    FTHADGIV.ADDRSLT2.OPR:=0
351 |           END;(*BB1 FALSE*)
352 |        END;(*A2 FALSE*)
353 |                 END(*A1*)
354 |               ELSE
355 |         BEGIN (*A1 FALSE*)
356 |          K:=COUNTICNUM(.J.);
357 |           IF (K<= MAXIC)THEN
358 |           BEGIN (*BB1*)
359 |            COUNTICNUM(.J.):=COUNTICNUM(.J.)+1;
360 |          IF(((CELLBLOCK(.J,K.).OPCODE)=SQURT)OR ((CELLBLOCK(.J,K.).OPCODE
361 |             =SQAR)))THEN
362 |              BEGIN(*BB2*)
363 |               IF((CELLBLOCK(.J,K.).OPRAND1 <> 99))THEN
364 |                BEGIN(*BB3*)
365 |                 IF(K<>FETCHREG(.J.))THEN
366 |                  BEGIN(*BB4*)
367 |                   FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
368 |                   FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
369 |                   FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
370 |                   FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
371 |                   FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
372 |                   FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
373 |                   FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
374 |                   FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
375 |                   FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
376 |                   FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
377 |                   FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
378 |                   FETCHREG(.J.):=K
379 |                  END(*BB4*)
380 |                 ELSE
381 |                  BEGIN(*BB4 FLASE*)
382 |                   FTHADGIV.OPCODE:=NOP;
383 |                   FTHADGIV.OPRAND1:=0;
384 |                   FTHADGIV.OPRAND2:=0;
385 |                   FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
```

```
                    FTHADGIV.ADDRSLT1.CB:=0;
                    FTHADGIV.ADDRSLT1.IC:=0;
                    FTHADGIV.ADDRSLT1.OPR:=0;
                    FTHADGIV.ADDRSLT2.CB:=0;
                    FTHADGIV.ADDRSLT2.IC:=0;
                    FTHADGIV.ADDRSLT2.OPR:=0
                  END;(*BB4 FALSE*)
                END(*BB3*)
               ELSE
                BEGIN(*BB3 FALSE*)
                    FTHADGIV.OPCODE:=NOP;
                    FTHADGIV.OPRAND1:=0;
                    FTHADGIV.OPRAND2:=0;
                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                    FTHADGIV.STAR:='A';
                    FTHADGIV.ADDRSLT1.CB:=0;
                    FTHADGIV.ADDRSLT1.IC:=0;
                    FTHADGIV.ADDRSLT1.OPR:=0;
                    FTHADGIV.ADDRSLT2.CB:=0;
                    FTHADGIV.ADDRSLT2.IC:=0;
                    FTHADGIV.ADDRSLT2.OPR:=0
                END;(*BB3 FALSE*)
              END(*BB2*)
             ELSE(*BB2 FALSE*)
              BEGIN(*BB2F1*)
            IF((CELLBLOCK(.J,K.).OPRAND1<>99)AND(CELLBLOCK(.J,K.).OPRAND2<>99)) THEN
              BEGIN(**BB2 F2*)
               IF(K<>FETCHREG(.J.))THEN
                BEGIN(**BB2 F3*)
                    FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
                    FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
                    FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                    FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
                    FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
                    FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
                    FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
                    FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
                    FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
                    FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
                    FETCHREG(.J.):=K
                END(*BB2 F3*)
               ELSE(*BB2 F3 FLASE*)
                BEGIN(*BB2 F3 FALSE*)
                    FTHADGIV.OPCODE:=NOP;
                    FTHADGIV.OPRAND1:=0;
                    FTHADGIV.OPRAND2:=0;
                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                    FTHADGIV.STAR:='A';
                    FTHADGIV.ADDRSLT1.CB:=0;
                    FTHADGIV.ADDRSLT1.IC:=0;
                    FTHADGIV.ADDRSLT1.OPR:=0;
                    FTHADGIV.ADDRSLT2.CB:=0;
                    FTHADGIV.ADDRSLT2.IC:=0;
                    FTHADGIV.ADDRSLT2.OPR:=0
                END;(*BB2 F3 FALSE*)
             END(*BB2 F2*)
            ELSE(*BB2 F2 FALSE*)
             BEGIN(*BB2 F2 FALSE*)
                    FTHADGIV.OPCODE:=NOP;
                    FTHADGIV.OPRAND1:=0;
                    FTHADGIV.OPRAND2:=0;
                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                    FTHADGIV.STAR:='A';
```

```
453                     FTHADGIV.ADDRSLT1.OPR:=0;
454                     FTHADGIV.ADDRSLT2.CB:=0;
455                     FTHADGIV.ADDRSLT2.IC:=0;
456                     FTHADGIV.ADDRSLT2.OPR:=0
457               END;(*BB2 F2 FALSE*)
458              END;(*BB2 F1*)
459             END(*BB1*)
460             ELSE(*BB1 FALSE*)
461              BEGIN(*BB1 FALSE*)
462                     FTHADGIV.OPCODE:=NOP;
463                     FTHADGIV.OPRAND1:=0;
464                     FTHADGIV.OPRAND2:=0;
465                     FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
466                     FTHADGIV.STAR:='A';
467                     FTHADGIV.ADDRSLT1.CB:=0;
468                     FTHADGIV.ADDRSLT1.IC:=0;
469                     FTHADGIV.ADDRSLT1.OPR:=0;
470                     FTHADGIV.ADDRSLT2.CB:=0;
471                     FTHADGIV.ADDRSLT2.IC:=0;
472                     FTHADGIV.ADDRSLT2.OPR:=0
473              END;(*BB1 FALSE*)
474             END;(*A1 FALSE*)
475                END(*AA*)
476             ELSE
477                BEGIN (*AA FALSE*)
478             IF((CELLBLOCK(.J,K.).OPRAND1<>99)AND(CELLBLOCK(.J,K.).OPRAND2<>99)) THEN
479             BEGIN(*AA1*)
480              IF(K<>FETCHREG(.J.))THEN
481               BEGIN(*AA2*)
482                     FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
483                     FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
484                     FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
485                     FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
486                     FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
487                     FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
488                     FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
489                     FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
490                     FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
491                     FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
492                     FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
493               FETCHREG(.J.):=K
494               END(**AA2*)
495              ELSE
496            BEGIN(*AA2 FALSE*)
497             K:=COUNTICNUM(.J.);
498              IF (K<= MAXIC)THEN
499               BEGIN (*BB1*)
500                COUNTICNUM(.J.):=COUNTICNUM(.J.)+1;
501              IF(((CELLBLOCK(.J,K.).OPCODE)=SQURT)OR ((CELLBLOCK(.J,K.).OPCODE
502                 =SQAR)))THEN
503                  BEGIN(*BB2*)
504                   IF((CELLBLOCK(.J,K.).OPRAND1 <> 99))THEN
505                    BEGIN(*BB3*)
506                     IF(K<>FETCHREG(.J.))THEN
507                      BEGIN(*BB4*)
508                       FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
509                       FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
510                       FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
511                       FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
512                       FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
513                       FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
514                       FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
515                       FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
```

```
517 |                    FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
518 |                    FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
519 |                    FETCHREG(.J.):=K
520 |                   END(*BB4*)
521 |                  ELSE
522 |                   BEGIN(*BB4 FLASE*)
523 |                    FTHADGIV.OPCODE:=NOP;
524 |                    FTHADGIV.OPRAND1:=0;
525 |                    FTHADGIV.OPRAND2:=0;
526 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
527 |                    FTHADGIV.STAR:='A';
528 |                    FTHADGIV.ADDRSLT1.CB:=0;
529 |                    FTHADGIV.ADDRSLT1.IC:=0;
530 |                    FTHADGIV.ADDRSLT1.OPR:=0;
531 |                    FTHADGIV.ADDRSLT2.CB:=0;
532 |                    FTHADGIV.ADDRSLT2.IC:=0;
533 |                    FTHADGIV.ADDRSLT2.OPR:=0
534 |                   END;(*BB4 FALSE*)
535 |                 END(*BB3*)
536 |                ELSE
537 |                 BEGIN(*BB3 FALSE*)
538 |                     FTHADGIV.OPCODE:=NOP;
539 |                     FTHADGIV.OPRAND1:=0;
540 |                     FTHADGIV.OPRAND2:=0;
541 |                     FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
542 |                     FTHADGIV.STAR:='A';
543 |                     FTHADGIV.ADDRSLT1.CB:=0;
544 |                     FTHADGIV.ADDRSLT1.IC:=0;
545 |                     FTHADGIV.ADDRSLT1.OPR:=0;
546 |                     FTHADGIV.ADDRSLT2.CB:=0;
547 |                     FTHADGIV.ADDRSLT2.IC:=0;
548 |                     FTHADGIV.ADDRSLT2.OPR:=0
549 |                  END;(*BB3 FALSE*)
550 |               END(*BB2*)
551 |              ELSE(*BB2 FALSE*)
552 |               BEGIN(*BB2F1*)
553 |          IF((CELLBLOCK(.J,K.).OPRAND1<>99)AND(CELLBLOCK(.J,K.).OPRAND2<>99)) THEN
554 |              BEGIN(**BB2 F2*)
555 |               IF(K<>FETCHREG(.J.))THEN
556 |                BEGIN(**BB2 F3*)
557 |                    FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
558 |                    FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
559 |                    FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
560 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
561 |                    FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
562 |                    FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
563 |                    FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
564 |                    FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
565 |                    FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
566 |                    FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
567 |                    FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
568 |                    FETCHREG(.J.):=K
569 |                END(*BB2 F3*)
570 |              ELSE(*BB2 F3 FLASE*)
571 |               BEGIN(*BB2 F3 FALSE*)
572 |                    FTHADGIV.OPCODE:=NOP;
573 |                    FTHADGIV.OPRAND1:=0;
574 |                    FTHADGIV.OPRAND2:=0;
575 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
576 |                    FTHADGIV.STAR:='A';
577 |                    FTHADGIV.ADDRSLT1.CB:=0;
578 |                    FTHADGIV.ADDRSLT1.IC:=0;
579 |                    FTHADGIV.ADDRSLT1.OPR:=0;
580 |                    FTHADGIV.ADDRSLT2.CB:=0;
```

```
582 |                 FTHADGIV.ADDRSLT2.OPR:=0
583 |               END;(*BB2 F3 FALSE*)
584 |             END(*BB2 F2*)
585 |            ELSE(*BB2 F2 FALSE*)
586 |             BEGIN(*BB2 F2 FALSE*)
587 |                    FTHADGIV.OPCODE:=NOP;
588 |                    FTHADGIV.OPRAND1:=0;
589 |                    FTHADGIV.OPRAND2:=0;
590 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
591 |                    FTHADGIV.STAR:='A';
592 |                    FTHADGIV.ADDRSLT1.CB:=0;
593 |                    FTHADGIV.ADDRSLT1.IC:=0;
594 |                    FTHADGIV.ADDRSLT1.OPR:=0;
595 |                    FTHADGIV.ADDRSLT2.CB:=0;
596 |                    FTHADGIV.ADDRSLT2.IC:=0;
597 |                    FTHADGIV.ADDRSLT2.OPR:=0
598 |             END;(*BB2 F2 FALSE*)
599 |            END;(*BB2 F1*)
600 |           END(*BB1*)
601 |           ELSE(*BB1 FALSE*)
602 |            BEGIN(*BB1 FALSE*)
603 |                    FTHADGIV.OPCODE:=NOP;
604 |                    FTHADGIV.OPRAND1:=0;
605 |                    FTHADGIV.OPRAND2:=0;
606 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
607 |                    FTHADGIV.STAR:='A';
608 |                    FTHADGIV.ADDRSLT1.CB:=0;
609 |                    FTHADGIV.ADDRSLT1.IC:=0;
610 |                    FTHADGIV.ADDRSLT1.OPR:=0;
611 |                    FTHADGIV.ADDRSLT2.CB:=0;
612 |                    FTHADGIV.ADDRSLT2.IC:=0;
613 |                    FTHADGIV.ADDRSLT2.OPR:=0
614 |             END;(*BB1 FALSE*)
615 |          END;(*AA2 FALSE*)
616 |         END(*AA1*)
617 |           ELSE
618 |         BEGIN(*AA1 FALSE*)
619 |          K:=COUNTICNUM(.J.);
620 |           IF (K<= MAXIC)THEN
621 |            BEGIN (*BB1*)
622 |             COUNTICNUM(.J.):=COUNTICNUM(.J.)+1;
623 |            IF(((CELLBLOCK(.J,K.).OPCODE)=SQURT)OR ((CELLBLOCK(.J,K.).OPCODE
624 |              =SQAR)))THEN
625 |               BEGIN(*BB2*)
626 |                IF((CELLBLOCK(.J,K.).OPRAND1 <> 99))THEN
627 |                 BEGIN(*BB3*)
628 |                  IF(K<>FETCHREG(.J.))THEN
629 |                   BEGIN(*BB4*)
630 |                    FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
631 |                    FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
632 |                    FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
633 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
634 |                    FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
635 |                    FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
636 |                    FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
637 |                    FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
638 |                    FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
639 |                    FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
640 |                    FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
641 |                    FETCHREG(.J.):=K
642 |                   END(*BB4*)
643 |                  ELSE
644 |                   BEGIN(*BB4 FLASE*)
645 |                    FTHADGIV.OPCODE:=NOP;
```

```
                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                    FTHADGIV.STAR:='A';
                    FTHADGIV.ADDRSLT1.CB:=0;
                    FTHADGIV.ADDRSLT1.IC:=0;
                    FTHADGIV.ADDRSLT1.OPR:=0;
                    FTHADGIV.ADDRSLT2.CB:=0;
                    FTHADGIV.ADDRSLT2.IC:=0;
                    FTHADGIV.ADDRSLT2.OPR:=0
                  END;(*BB4 FALSE*)
                END(*BB3*)
               ELSE
                BEGIN(*BB3 FALSE*)
                    FTHADGIV.OPCODE:=NOP;
                    FTHADGIV.OPRAND1:=0;
                    FTHADGIV.OPRAND2:=0;
                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                    FTHADGIV.STAR:='A';
                    FTHADGIV.ADDRSLT1.CB:=0;
                    FTHADGIV.ADDRSLT1.IC:=0;
                    FTHADGIV.ADDRSLT1.OPR:=0;
                    FTHADGIV.ADDRSLT2.CB:=0;
                    FTHADGIV.ADDRSLT2.IC:=0;
                    FTHADGIV.ADDRSLT2.OPR:=0
                END;(*BB3 FALSE*)
            END(*BB2*)
           ELSE(*BB2 FALSE*)
            BEGIN(*BB2F1*)
      IF((CELLBLOCK(.J,K.).OPRAND1<>99)AND(CELLBLOCK(.J,K.).OPRAND2<>99)) THEN
            BEGIN(**BB2 F2*)
             IF(K<>FETCHREG(.J.))THEN
              BEGIN(**BB2 F3*)
                  FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
                  FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
                  FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
                  FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                  FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
                  FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
                  FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
                  FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
                  FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
                  FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
                  FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
                  FETCHREG(.J.):=K
              END(*BB2 F3*)
             ELSE(*BB2 F3 FLASE*)
              BEGIN(*BB2 F3 FALSE*)
                  FTHADGIV.OPCODE:=NOP;
                  FTHADGIV.OPRAND1:=0;
                  FTHADGIV.OPRAND2:=0;
                  FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                  FTHADGIV.STAR:='A';
                  FTHADGIV.ADDRSLT1.CB:=0;
                  FTHADGIV.ADDRSLT1.IC:=0;
                  FTHADGIV.ADDRSLT1.OPR:=0;
                  FTHADGIV.ADDRSLT2.CB:=0;
                  FTHADGIV.ADDRSLT2.IC:=0;
                  FTHADGIV.ADDRSLT2.OPR:=0
              END;(*BB2 F3 FALSE*)
            END(*BB2 F2*)
           ELSE(*BB2 F2 FALSE*)
            BEGIN(*BB2 F2 FALSE*)
                  FTHADGIV.OPCODE:=NOP;
                  FTHADGIV.OPRAND1:=0;
```

```
                    FTHADGIV.STAR:='A';
                    FTHADGIV.ADDRSLT1.CB:=0;
                    FTHADGIV.ADDRSLT1.IC:=0;
                    FTHADGIV.ADDRSLT1.OPR:=0;
                    FTHADGIV.ADDRSLT2.CB:=0;
                    FTHADGIV.ADDRSLT2.IC:=0;
                    FTHADGIV.ADDRSLT2.OPR:=0
            END;(*BB2 F2 FALSE*)
          END;(*BB2 F1*)
        END(*BB1*)
        ELSE(*BB1 FALSE*)
         BEGIN(*BB1 FALSE*)
                    FTHADGIV.OPCODE:=NOP;
                    FTHADGIV.OPRAND1:=0;
                    FTHADGIV.OPRAND2:=0;
                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                    FTHADGIV.STAR:='A';
                    FTHADGIV.ADDRSLT1.CB:=0;
                    FTHADGIV.ADDRSLT1.IC:=0;
                    FTHADGIV.ADDRSLT1.OPR:=0;
                    FTHADGIV.ADDRSLT2.CB:=0;
                    FTHADGIV.ADDRSLT2.IC:=0;
                    FTHADGIV.ADDRSLT2.OPR:=0
          END;(*BB1 FALSE*)
       END;(*AA1 FALSE*)
           END;(*AA FALSE*)
      END;(*NOT EQ FIN*)
    END(*TRUE UPDATE YES*)
   ELSE(*IF NOT UPDATEYES TRUE*)
    BEGIN(*BB*)
     K:=COUNTICNUM(.J.);
      IF (K<= MAXIC)THEN
       BEGIN (*BB1*)
        COUNTICNUM(.J.):=COUNTICNUM(.J.)+1;
      IF(((CELLBLOCK(.J,K.).OPCODE)=SQURT)OR ((CELLBLOCK(.J,K.).OPCODE
        =SQAR)))THEN
         BEGIN(*BB2*)
          IF((CELLBLOCK(.J,K.).OPRAND1 <> 99))THEN
           BEGIN(*BB3*)
            IF(K<>FETCHREG(.J.))THEN
             BEGIN(*BB4*)
              FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
              FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
              FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
              FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
              FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
              FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
              FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
              FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
              FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
              FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
              FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
              FETCHREG(.J.):=K
             END(*BB4*)
            ELSE
             BEGIN(*BB4 FLASE*)
              FTHADGIV.OPCODE:=NOP;
              FTHADGIV.OPRAND1:=0;
              FTHADGIV.OPRAND2:=0;
              FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
              FTHADGIV.STAR:='A';
              FTHADGIV.ADDRSLT1.CB:=0;
              FTHADGIV.ADDRSLT1.IC:=0;
```

```
                    FTHADGIV.ADDRSLT2.CB:=0;
                    FTHADGIV.ADDRSLT2.IC:=0;
                    FTHADGIV.ADDRSLT2.OPR:=0
                  END;(*BB4 FALSE*)
                END(*BB3*)
               ELSE
                BEGIN(*BB3 FALSE*)
                    FTHADGIV.OPCODE:=NOP;
                    FTHADGIV.OPRAND1:=0;
                    FTHADGIV.OPRAND2:=0;
                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                    FTHADGIV.STAR:='A';
                    FTHADGIV.ADDRSLT1.CB:=0;
                    FTHADGIV.ADDRSLT1.IC:=0;
                    FTHADGIV.ADDRSLT1.OPR:=0;
                    FTHADGIV.ADDRSLT2.CB:=0;
                    FTHADGIV.ADDRSLT2.IC:=0;
                    FTHADGIV.ADDRSLT2.OPR:=0
                END;(*BB3 FALSE*)
            END(*BB2*)
           ELSE(*BB2 FALSE*)
            BEGIN(*BB2F1*)
      IF((CELLBLOCK(.J,K.).OPRAND1<>99)AND(CELLBLOCK(.J,K.).OPRAND2<>99)) THEN
            BEGIN(**BB2 F2*)
             IF(K<>FETCHREG(.J.))THEN
              BEGIN(**BB2 F3*)
                  FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
                  FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
                  FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
                  FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                  FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
                  FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
                  FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
                  FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
                  FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
                  FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
                  FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
                  FETCHREG(.J.):=K
              END(*BB2 F3*)
             ELSE(*BB2 F3 FLASE*)
              BEGIN(*BB2 F3 FALSE*)
                  FTHADGIV.OPCODE:=NOP;
                  FTHADGIV.OPRAND1:=0;
                  FTHADGIV.OPRAND2:=0;
                  FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                  FTHADGIV.STAR:='A';
                  FTHADGIV.ADDRSLT1.CB:=0;
                  FTHADGIV.ADDRSLT1.IC:=0;
                  FTHADGIV.ADDRSLT1.OPR:=0;
                  FTHADGIV.ADDRSLT2.CB:=0;
                  FTHADGIV.ADDRSLT2.IC:=0;
                  FTHADGIV.ADDRSLT2.OPR:=0
              END;(*BB2 F3 FALSE*)
          END(*BB2 F2*)
         ELSE(*BB2 F2 FALSE*)
          BEGIN(*BB2 F2 FALSE*)
                  FTHADGIV.OPCODE:=NOP;
                  FTHADGIV.OPRAND1:=0;
                  FTHADGIV.OPRAND2:=0;
                  FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                  FTHADGIV.STAR:='A';
                  FTHADGIV.ADDRSLT1.CB:=0;
                  FTHADGIV.ADDRSLT1.IC:=0;
                  FTHADGIV.ADDRSLT1.OPR:=0;
```

```
842 |                    FTHADGIV.ADDRSLT2.IC:=0;
843 |                    FTHADGIV.ADDRSLT2.OPR:=0
844 |            END;(*BB2 F2 FALSE*)
845 |          END;(*BB2 F1*)
846 |         END(*BB1*)
847 |         ELSE(*BB1 FALSE*)
848 |          BEGIN(*BB1 FALSE*)
849 |                    FTHADGIV.OPCODE:=NOP;
850 |                    FTHADGIV.OPRAND1:=0;
851 |                    FTHADGIV.OPRAND2:=0;
852 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
853 |                    FTHADGIV.STAR:='A';
854 |                    FTHADGIV.ADDRSLT1.CB:=0;
855 |                    FTHADGIV.ADDRSLT1.IC:=0;
856 |                    FTHADGIV.ADDRSLT1.OPR:=0;
857 |                    FTHADGIV.ADDRSLT2.CB:=0;
858 |                    FTHADGIV.ADDRSLT2.IC:=0;
859 |                    FTHADGIV.ADDRSLT2.OPR:=0
860 |           END;(*BB1 FALSE*)
861 |         END;(*BB*)
862 |       END;(*PROCEDURE FETCHUNIT*)
863 |  END;(*111*)
864 |  (**********************PROCEDURE FOR PROCESSING ELEMENT**********)
865 |  (**************ALL VARIABLES ARE GLOBAL*******************)
866 |  (**************ISSUE:CODING OF BR:-SINCE BR IS INSIDE THE INTEGER*)
867 |  (*REPRESENTATION 0 0 0 0 0 BR THEREFORE BR=0 IS REPRESENTED AS   *)
868 |  (**********BR=0=>999, BR=1=>9999************************************)
869 |  PROCEDURE PROCELEM;
870 |  (*LOCAL VARIABLES,STORE,STOR,VARX=OPRAND1,VARY=OPRAND2,VARZ=RESULT*)
871 |  CONST
872 |   CON1=999;
873 |   CON2=9999;
874 |  VAR
875 |   VOPCOD:CODE;
876 |   STORE:STARVAL;
877 |   VARX,VARY,VARZ,PIPINDX,STOR:INTEGER;
878 |  BEGIN
879 |     VARX:=FTHADGIV.OPRAND1;
880 |     VARY:=FTHADGIV.OPRAND2;
881 |     VOPCOD:=FTHADGIV.OPCODE;
882 |       CASE VOPCOD OF
883 |         NOP:
884 |             VARZ:=0;
885 |         ADD:
886 |             VARZ:=VARX+VARY;
887 |         SUB:
888 |             VARZ:=VARX-VARY;
889 |             (***NOTE OPRAND1-OPRAND2**)
890 |         MUL:
891 |             VARZ:=VARX*VARY;
892 |         DIVI:
893 |             VARZ:=VARX DIV VARY;
894 |  (******NOTE:SINGLE VALUE OPERATIONS WILL TAKE OPRAND1********)
895 |         SQURT:
896 |             VARZ:=ROUND(SQRT(VARX));
897 |         SQAR:
898 |             VARZ:=SQR(VARX);
899 |         COMP:
900 |             IF (VARX=VARY)THEN
901 |                 VARZ:=CON1 (*I.E OPR1=FINVAL,BR=0*)
902 |              ELSE
903 |                 VARZ:=CON2;(*IE OPR1 NOT EQUAL TO FINVAL,BR=1*)
904 |         LOOPI:
905 |               BEGIN(*JUST FOLLOWING SIMU. BREAKDOWN*)
```

```
907   |  (*RESULT PACKET BEING SENT TO THE DISTRIBUTION NETWORK**)
908   |                  VARZ:=VARY+0;
909   |                  IF (VARX=999)THEN FTHADGIV.STAR:='C';
910   |                  IF (VARX=9999)THEN FTHADGIV.STAR:='D'
911   |                END;
912   |             FIN:
913   |                  VARZ:=0
914   |       END;(*CASE*)
915   |  (********PUTTING THE RESULT IN THE ARRAY AND FORMING THE ***)
916   |  (*RESULT PACKET***************)
917   |  (***PIPELINE SIMULATION*OF RESULT*******************)
918   |       FOR PIPINDX:=1 TO PIPELENGTH DO
919   |         BEGIN
920   |          STOR:=PROCCALC(.J,PIPINDX.).RESULT;
921   |          PROCCALC(.J,PIPINDX.).RESULT:=VARZ;
922   |          VARZ:=STOR
923   |         END;
924   |  (*REMEMBER WHY USING J:BECAUSE J IS COMING FROM THE FETCH UNIT*)
925   |  (* BECAUSE RIGHT AWAY AFTER FETCH UNIT THE PR.ELEMTNET IS CALLED*)
926   |  (*REMEMBER THE 1TO 1 CORR TO CB AND PE**)
927   |  (* NOTICE THAT THE J REPRESENTS THE CELLBLOCK****)
928   |  (*****************************************************************)
929   |  (**********NOW SIMULATING THE ADDRESS PIPELING**************)
930   |  (*************WARNING WATCH OUT FOR THE J************)
931   |  (**********B E W A R E*********************************)
932   |       FOR PIPINDX:=1 TO PIPELENGTH DO
933   |         BEGIN
934   |          STOR:=PROCADDRESS(.J,PIPINDX.).ADDRSLT1.CB;
935   |          PROCADDRESS(.J,PIPINDX.).ADDRSLT1.CB:=FTHADGIV.ADDRSLT1.CB;
936   |          FTHADGIV.ADDRSLT1.CB:=STOR
937   |         END;
938   |  (******************NEXT ADDRESS FIELD PIPELINE*****************)
939   |       FOR PIPINDX:=1 TO PIPELENGTH DO
940   |         BEGIN
941   |          STOR:=PROCADDRESS(.J,PIPINDX.).ADDRSLT1.IC;
942   |          PROCADDRESS(.J,PIPINDX.).ADDRSLT1.IC:=FTHADGIV.ADDRSLT1.IC;
943   |          FTHADGIV.ADDRSLT1.IC:=STOR
944   |         END;
945   |  (*****************************************************************)
946   |       FOR PIPINDX:=1 TO PIPELENGTH DO
947   |         BEGIN
948   |          STOR:=PROCADDRESS(.J,PIPINDX.).ADDRSLT1.OPR;
949   |          PROCADDRESS(.J,PIPINDX.).ADDRSLT1.OPR:=FTHADGIV.ADDRSLT1.OPR;
950   |          FTHADGIV.ADDRSLT1.OPR:=STOR
951   |         END;
952   |  (*****************************************************************)
953   |       FOR PIPINDX:=1 TO PIPELENGTH DO
954   |         BEGIN
955   |          STOR:=PROCADDRESS(.J,PIPINDX.).ADDRSLT2.CB;
956   |          PROCADDRESS(.J,PIPINDX.).ADDRSLT2.CB:=FTHADGIV.ADDRSLT2.CB;
957   |          FTHADGIV.ADDRSLT2.CB:=STOR
958   |         END;
959   |  (******************NEXT ADDRESS FIELD PIPELINE*****************)
960   |       FOR PIPINDX:=1 TO PIPELENGTH DO
961   |         BEGIN
962   |          STOR:=PROCADDRESS(.J,PIPINDX.).ADDRSLT2.IC;
963   |          PROCADDRESS(.J,PIPINDX.).ADDRSLT2.IC:=FTHADGIV.ADDRSLT2.IC;
964   |          FTHADGIV.ADDRSLT2.IC:=STOR
965   |         END;
966   |  (*****************************************************************)
967   |       FOR PIPINDX:=1 TO PIPELENGTH DO
968   |         BEGIN
969   |          STOR:=PROCADDRESS(.J,PIPINDX.).ADDRSLT2.OPR;
970   |          PROCADDRESS(.J,PIPINDX.).ADDRSLT2.OPR:=FTHADGIV.ADDRSLT2.OPR;
```

```
            END;
  (**********************NOW THE STAR FIELD PIPLINE*********************)
        FOR PIPINDX:=1 TO PIPELENGTH DO
          BEGIN
           STORE:=PROCADDRESS(.J,PIPINDX.).STAR;
           PROCADDRESS(.J,PIPINDX.).STAR:=FTHADGIV.STAR;
           FTHADGIV.STAR:=STORE
          END;
  (*************************************************************************)
  (*******************FORMING THE RESULT PACKET*************************)
        RESLTPAC.RESULT:=PROCCALC(.J,PIPELENGTH.).RESULT;
        RESLTPAC.STAR:=PROCADDRESS(.J,PIPELENGTH.).STAR;
        RESLTPAC.ADDRSLT1.CB:=PROCADDRESS(.J,PIPELENGTH.).ADDRSLT1.CB;
        RESLTPAC.ADDRSLT1.IC:=PROCADDRESS(.J,PIPELENGTH.).ADDRSLT1.IC;
        RESLTPAC.ADDRSLT1.OPR:=PROCADDRESS(.J,PIPELENGTH.).ADDRSLT1.OPR;
        RESLTPAC.ADDRSLT2.CB:=PROCADDRESS(.J,PIPELENGTH.).ADDRSLT2.CB;
        RESLTPAC.ADDRSLT2.IC:=PROCADDRESS(.J,PIPELENGTH.).ADDRSLT2.IC;
        RESLTPAC.ADDRSLT2.OPR:=PROCADDRESS(.J,PIPELENGTH.).ADDRSLT2.OPR
  END;(*PROCESSING ELEMENT*)
  (*************************************************************************)
  (*************************************************************************)
  (*****************************PROCEDURE DISTRIBUTION NETWORK******)
  PROCEDURE DISTNET;
  (************************LOCAL VARIABLE DEFINED****************)
  VAR
    LSTAR:STARVAL;
  BEGIN
  (****STAR NEEDS TO BE TRANSMITTED AS IT IS TO THE UPDATE UNIT SO***)
     DISADDGIV.STAR:=RESLTPAC.STAR;
  (*********************SAME WITH RESULT*********************)
      DISADDGIV.RESULT:=RESLTPAC.RESULT;
      LSTAR:=RESLTPAC.STAR;
        CASE LSTAR OF
         'A':(* *=00,EXAMINE ADDRSLT1 TO SEND TO CORR.CB*)
             BEGIN
               DISADDGIV.ADDRSLT1.CB:=RESLTPAC.ADDRSLT1.CB;
               DISADDGIV.ADDRSLT1.IC:=RESLTPAC.ADDRSLT1.IC;
               DISADDGIV.ADDRSLT1.OPR:=RESLTPAC.ADDRSLT1.OPR;
  (***************PUTTING THE ADDRSLT2 TO BE 0*****************)
               DISADDGIV.ADDRSLT2.CB:=0;
               DISADDGIV.ADDRSLT2.IC:=0;
               DISADDGIV.ADDRSLT2.OPR:=0
             END;
         'B':(* *=01,BOTH ADDRSLT1 AND ADDRSLT2 ARE VALID*)
             (**BOTH NEED TO BE SEND TO CORROSPONDING CB AND THEN*)
             (*TO THE UPDATE UNIT. UPDATE UNIT WILL POINT THE CONFLICT*)
             (* IF THE UPDATING IS DONE WITHIN THE SAME CELL BLOCK**)
             BEGIN
               DISADDGIV.ADDRSLT1.CB:=RESLTPAC.ADDRSLT1.CB;
               DISADDGIV.ADDRSLT1.IC:=RESLTPAC.ADDRSLT1.IC;
               DISADDGIV.ADDRSLT1.OPR:=RESLTPAC.ADDRSLT1.OPR;
               DISADDGIV.ADDRSLT2.CB:=RESLTPAC.ADDRSLT2.CB;
               DISADDGIV.ADDRSLT2.IC:=RESLTPAC.ADDRSLT2.IC;
               DISADDGIV.ADDRSLT2.OPR:=RESLTPAC.ADDRSLT2.OPR
             END;
         'C':(* *=10 I.E TAKE ADDRESS RESLULT 1 TO CORR. CB*)
             (****I.E  SAME AS 'A'**,THIS WILL BE USED FOR LOOP USING BR*)
             BEGIN
               DISADDGIV.ADDRSLT1.CB:=RESLTPAC.ADDRSLT1.CB;
               DISADDGIV.ADDRSLT1.IC:=RESLTPAC.ADDRSLT1.IC;
               DISADDGIV.ADDRSLT1.OPR:=RESLTPAC.ADDRSLT1.OPR;
  (***************PUTTING THE ADDRSLT2 TO BE 0*****************)
               DISADDGIV.ADDRSLT2.CB:=0;
               DISADDGIV.ADDRSLT2.IC:=0;
```

```
1037                END;
1038          'D':(** *=11,THIS USED FOR LOOPING USING BR*)
1039                BEGIN
1040                  DISADDGIV.ADDRSLT2.CB:=RESLTPAC.ADDRSLT1.CB;
1041                  DISADDGIV.ADDRSLT2.IC:=RESLTPAC.ADDRSLT1.IC;
1042                  DISADDGIV.ADDRSLT2.OPR:=RESLTPAC.ADDRSLT1.OPR;
1043    (****************PUTTING THE ADDRSLT2 TO BE 0*****************)
1044                  DISADDGIV.ADDRSLT1.CB:=0;
1045                  DISADDGIV.ADDRSLT1.IC:=0;
1046                  DISADDGIV.ADDRSLT1.OPR:=0
1047                END
1048             END(*CASE*)
1049    END;(* END PROCEDURE DISTRIBUTION NETWORK*)
1050    (*****************************************************************)
1051    (**THE ONLY UNIT WHICH CAN WRITE IN THE MEMORY, THE ADDRESS IS *)
1052    (*GIVEN ONLY OF THE IC DEPENDING ON CB ALREADY FOUND BY THE DIS.NETW*)
1053    PROCEDURE UPDATE;
1054    VAR
1055      LLSTAR:STARVAL;
1056      VK,VI,VL,STOR,VARJ,VARK,VOPR,VARX,VGG:INTEGER;
1057      VB,VSTOR:BOOLEAN;
1058    BEGIN(*NOTICE YOU CAN REPLACE LOTS OF CODE BY SINGLE PROC. BUT IT*)
1059         (*IS AVOIDED BECAUSE TO REPRESENT THE FUNCTIONALITY OF UNITS*)
1060      LLSTAR:=DISADDGIV.STAR;
1061        CASE LLSTAR OF
1062         'A':
1063            BEGIN
1064             VK:=DISADDGIV.ADDRSLT1.CB;
1065             IF
1066              (((DISADDGIV.ADDRSLT1.CB)=0)AND((DISADDGIV.ADDRSLT1.IC)=0)AND
1067              ((DISADDGIV.ADDRSLT1.OPR)=0))THEN
1068    (*************************************************************)
1069               BEGIN(*FALSE*)
1070                 IF SIMSAVB(.CBLOOPVAL.) THEN
1071                  BEGIN(*2*)
1072                   VGG:=SIMSAVE(.CBLOOPVAL.);
1073                   FOR VI:=1 TO MAXCELBLK DO
1074                    BEGIN
1075                     VSTOR:=UPDATYES(.VGG,VI.);
1076                     UPDATYES(.VGG,VI.):=SIMSAVB(.CBLOOPVAL.);
1077                     SIMSAVB(.CBLOOPVAL.):=VSTOR
1078                    END;
1079                   FOR VI:=1 TO MAXCELBLK DO
1080                    BEGIN
1081                     STOR:=UPADDGIV(.VGG,VI.).IC;
1082                     UPADDGIV(.VGG,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1083                     SIMSAVE(.CBLOOPVAL.):=STOR
1084                    END;
1085                    SIMSAVE(.CBLOOPVAL.):=0;
1086                    SIMSAVB(.CBLOOPVAL.):=FALSE
1087                  END;(*2*)
1088                VB:=FALSE;
1089                FOR VI:= 1 TO MAXCELBLK DO
1090                 BEGIN(*1*)
1091                  VSTOR:=UPDATYES(.CBLOOPVAL,VI.);
1092                  UPDATYES(.CBLOOPVAL,VI.):=VB;
1093                  VB:=VSTOR
1094                 END;(*1*)
1095                  VARX:=0;
1096                  FOR VI:=1 TO MAXCELBLK DO
1097                   BEGIN
1098                    STOR:=UPADDGIV(.CBLOOPVAL,VI.).IC;
1099                    UPADDGIV(.CBLOOPVAL,VI.).IC:=VARX;
1100                    VARX:=STOR
```

```
1102 |              END(*FALSE*)
1103 |            ELSE
1104 |             BEGIN(*1A*)
1105 |             (*GIVING THE ADDRESS VALUE TO THE FETHC UNIT,FOR*)
1106 |             (*THE INSTRUCTION JUST UPDATED*)
1107 |              IF (VK > CBLOOPVAL) THEN(*BECAUSE FIRST TIME AROUND IT WOULD*)
1108 |                                       (*BE FALSE**************************)
1109 |               BEGIN(*1*)
1110 |                IF SIMSAVB(.CBLOOPVAL.) THEN
1111 |                 BEGIN(*2*)
1112 |                  FOR VI:=1 TO MAXCELBLK DO
1113 |                   BEGIN
1114 |                    VSTOR:=UPDATYES(.VK,VI.);
1115 |                    UPDATYES(.VK,VI.):=SIMSAVB(.CBLOOPVAL.);
1116 |                    SIMSAVB(.CBLOOPVAL.):=VSTOR
1117 |                   END;
1118 |                  FOR VI:=1 TO MAXCELBLK DO
1119 |                   BEGIN
1120 |                    STOR:=UPADDGIV(.VK,VI.).IC;
1121 |                    UPADDGIV(.VK,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1122 |                    SIMSAVE(.CBLOOPVAL.):=STOR
1123 |                   END;
1124 |                  SIMSAVE(.CBLOOPVAL.):=VK;(*STORING FOR NEXT USE*)
1125 |                  SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1126 |                 END(*2*)
1127 |               ELSE(*IF SIMSAVB(.CBLOOPVAL.) IS FALSE*)
1128 |                (*JUST STORE FOR THE NEXT VALUE*)
1129 |                  SIMSAVE(.CBLOOPVAL.):=VK;(*STORING FOR NEXT USE*)
1130 |                  SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1131 |           END(*1*)
1132 |         ELSE(*IF VK<= TO CBLOOP VAL*)
1133 |          BEGIN(*3*)
1134 |                IF SIMSAVB(.CBLOOPVAL.) THEN
1135 |                 BEGIN(*2*)
1136 |                  VGG:=SIMSAVE(.CBLOOPVAL.);
1137 |                  FOR VI:=1 TO MAXCELBLK DO
1138 |                   BEGIN
1139 |                    VSTOR:=UPDATYES(.VGG,VI.);
1140 |                    UPDATYES(.VGG,VI.):=SIMSAVB(.CBLOOPVAL.);
1141 |                    SIMSAVB(.CBLOOPVAL.):=VSTOR
1142 |                   END;
1143 |                  FOR VI:=1 TO MAXCELBLK DO
1144 |                   BEGIN
1145 |                    STOR:=UPADDGIV(.VGG,VI.).IC;
1146 |                    UPADDGIV(.VGG,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1147 |                    SIMSAVE(.CBLOOPVAL.):=STOR
1148 |                   END;
1149 |                  SIMSAVE(.CBLOOPVAL.):=0;
1150 |                  SIMSAVB(.CBLOOPVAL.):=FALSE
1151 |                 END;(*2*)
1152 |               VB:=TRUE;
1153 |               FOR VI:=1 TO MAXCELBLK DO
1154 |                BEGIN
1155 |                 VSTOR:=UPDATYES(.VK,VI.);
1156 |                 UPDATYES(.VK,VI.):=VB;
1157 |                 VB:=VSTOR
1158 |                END;
1159 |               VARX:=DISADDGIV.ADDRSLT1.IC;
1160 |                FOR VI:=1 TO MAXCELBLK DO
1161 |                  BEGIN(*SAVING THE ADDRESS UPDATED BY UDU FOR F.U*)
1162 |                   STOR:=UPADDGIV(.VK,VI.).IC;
1163 |                   UPADDGIV(.VK,VI.).IC:=VARX;
1164 |                   VARX:=STOR
1165 |                  END;
```

```
1167 |  (*****************NEXT UPDATING THE CELL BLOCK*****************)
1168 |  (**************ACTUALLY WRITTING IN THE ARRY*WITH RESULT******)
1169 |              VOPR:=DISADDGIV.ADDRSLT1.OPR;
1170 |              VARJ:=DISADDGIV.ADDRSLT1.CB;
1171 |              VARK:=DISADDGIV.ADDRSLT1.IC;
1172 |               IF (VOPR=1)THEN
1173 |                CELLBLOCK(.VARJ,VARK.).OPRAND1:=DISADDGIV.RESULT;
1174 |               IF (VOPR=2)THEN
1175 |                CELLBLOCK(.VARJ,VARK.).OPRAND2:=DISADDGIV.RESULT
1176 |          END(*1A*)
1177 |         END;(*END BEGIN*)
1178 |         'B':
1179 |
1180 |           BEGIN
1181 |            IF
1182 |            ((((DISADDGIV.ADDRSLT1.CB)=0)AND((DISADDGIV.ADDRSLT1.IC)=0)AND
1183 |             ((DISADDGIV.ADDRSLT1.OPR)=0))AND
1184 |             (((DISADDGIV.ADDRSLT2.CB)=0)AND((DISADDGIV.ADDRSLT2.IC)=0)AND
1185 |             ((DISADDGIV.ADDRSLT2.OPR)=0)))THEN
1186 |  (*************************************************************)
1187 |             BEGIN(*FALSE *)
1188 |                IF SIMSAVB(.CBLOOPVAL.) THEN
1189 |                BEGIN(*2*)
1190 |                  VGG:=SIMSAVE(.CBLOOPVAL.);
1191 |                  FOR VI:=1 TO MAXCELBLK DO
1192 |                   BEGIN
1193 |                    VSTOR:=UPDATYES(.VGG,VI.);
1194 |                    UPDATYES(.VGG,VI.):=SIMSAVB(.CBLOOPVAL.);
1195 |                    SIMSAVB(.CBLOOPVAL.):=VSTOR
1196 |                   END;
1197 |                  FOR VI:=1 TO MAXCELBLK DO
1198 |                   BEGIN
1199 |                    STOR:=UPADDGIV(.VGG,VI.).IC;
1200 |                    UPADDGIV(.VGG,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1201 |                    SIMSAVE(.CBLOOPVAL.):=STOR
1202 |                   END;
1203 |                   SIMSAVE(.CBLOOPVAL.):=0;
1204 |                   SIMSAVB(.CBLOOPVAL.):=FALSE
1205 |                END;(*2*)
1206 |              VB:=FALSE;
1207 |              FOR VI:= 1 TO MAXCELBLK DO
1208 |               BEGIN(*1*)
1209 |                VSTOR:=UPDATYES(.CBLOOPVAL,VI.);
1210 |                UPDATYES(.CBLOOPVAL,VI.):=VB;
1211 |                VB:=VSTOR
1212 |               END;(*1*)
1213 |                VARX:=0;
1214 |                FOR VI:=1 TO MAXCELBLK DO
1215 |                 BEGIN
1216 |                  STOR:=UPADDGIV(.CBLOOPVAL,VI.).IC;
1217 |                  UPADDGIV(.CBLOOPVAL,VI.).IC:=VARX;
1218 |                  VARX:=STOR
1219 |                 END;
1220 |            END(*FALSE*)
1221 |           ELSE
1222 |        BEGIN(*1A*)
1223 |           (*GIVING THE ADDRESS VALUE TO THE FETHC UNIT,FOR*)
1224 |                 (*THE INSTRUCTION JUST UPDATED*)
1225 |  (*******ALTHOUGH BOTH THE INSTRUCTIONS WILL BE UPDATED I.E WRITTEN*)
1226 |  (**ON BUT TO THE FETCH UNIT ONLY ONE ADDRESS WILL BE SENT**I.E ADDRESLT1*)
1227 |             VK:=DISADDGIV.ADDRSLT1.CB;
1228 |             IF (VK > CBLOOPVAL) THEN(*BECAUSE FIRST TIME AROUND IT WOULD*)
1229 |                                      (*BE FALSE**************************)
1230 |              BEGIN(*1*)
```

```
1232 |                BEGIN(*2*)
1233 |                 FOR VI:=1 TO MAXCELBLK DO
1234 |                  BEGIN
1235 |                   VSTOR:=UPDATYES(.VK,VI.);
1236 |                   UPDATYES(.VK,VI.):=SIMSAVB(.CBLOOPVAL.);
1237 |                   SIMSAVB(.CBLOOPVAL.):=VSTOR
1238 |                  END;
1239 |                 FOR VI:=1 TO MAXCELBLK DO
1240 |                  BEGIN
1241 |                   STOR:=UPADDGIV(.VK,VI.).IC;
1242 |                   UPADDGIV(.VK,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1243 |                   SIMSAVE(.CBLOOPVAL.):=STOR
1244 |                  END;
1245 |                 SIMSAVE(.CBLOOPVAL.):=VK;(*STORING FOR NEXT USE*)
1246 |                 SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1247 |               END(*2*)
1248 |             ELSE(*IF SIMSAVB(.CBLOOPVAL.) IS FALSE*)
1249 |              (*JUST STORE FOR THE NEXT VALUE*)
1250 |                 SIMSAVE(.CBLOOPVAL.):=VK;(*STORING FOR NEXT USE*)
1251 |                 SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1252 |           END(*1*)
1253 |         ELSE(*IF VK<= TO CBLOOP VAL*)
1254 |          BEGIN(*3*)
1255 |                IF SIMSAVB(.CBLOOPVAL.) THEN
1256 |                 BEGIN(*2*)
1257 |                  VGG:=SIMSAVE(.CBLOOPVAL.);
1258 |                  FOR VI:=1 TO MAXCELBLK DO
1259 |                   BEGIN
1260 |                    VSTOR:=UPDATYES(.VGG,VI.);
1261 |                    UPDATYES(.VGG,VI.):=SIMSAVB(.CBLOOPVAL.);
1262 |                    SIMSAVB(.CBLOOPVAL.):=VSTOR
1263 |                   END;
1264 |                  FOR VI:=1 TO MAXCELBLK DO
1265 |                   BEGIN
1266 |                    STOR:=UPADDGIV(.VGG,VI.).IC;
1267 |                    UPADDGIV(.VGG,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1268 |                    SIMSAVE(.CBLOOPVAL.):=STOR
1269 |                   END;
1270 |                   SIMSAVE(.CBLOOPVAL.):=0;
1271 |                   SIMSAVB(.CBLOOPVAL.):=FALSE
1272 |                 END;(*2*)
1273 |              VB:=TRUE;
1274 |              FOR VI:=1 TO MAXCELBLK DO
1275 |               BEGIN
1276 |                VSTOR:=UPDATYES(.VK,VI.);
1277 |                UPDATYES(.VK,VI.):=VB;
1278 |                VB:=VSTOR
1279 |               END;
1280 |              VARX:=DISADDGIV.ADDRSLT1.IC;
1281 |               FOR VI:=1 TO MAXCELBLK DO
1282 |                BEGIN
1283 |                 STOR:=UPADDGIV(.VK,VI.).IC;
1284 |                 UPADDGIV(.VK,VI.).IC:=VARX;
1285 |                 VARX:=STOR
1286 |                END;
1287 |           END;(*3*)
1288 |             (**ADDRSLT2*)
1289 |              VL:=DISADDGIV.ADDRSLT2.CB;
1290 |             (*GIVING THE ADDRESS VALUE TO THE FETHC UNIT,FOR*)
1291 |             (*THE INSTRUCTION JUST UPDATED*)
1292 |       IF (VL=VK)THEN
1293 |        BEGIN(*VL=VK*)
1294 |              VB:=TRUE;
```

```
1297 |                VSTOR:=UPDATYES(.VL,VI.);
1298 |                UPDATYES(.VL,VI.):=VB;
1299 |                VB:=VSTOR
1300 |               END;
1301 |              VARX:=DISADDGIV.ADDRSLT2.IC;
1302 |               FOR VI:=1 TO MAXCELBLK DO
1303 |                BEGIN
1304 |                 STOR:=UPADDGIV(.VL,VI.).IC;
1305 |                 UPADDGIV(.VL,VI.).IC:=VARX;
1306 |                 VARX:=STOR
1307 |                END;
1308 |         END(*VL=VK*)
1309 |        ELSE
1310 |         BEGIN(*VL NOT EQUAL VK*)
1311 |               IF (VL > CBLOOPVAL) THEN(*BECAUSE FIRST TIME AROUND IT WOULD*)
1312 |                                        (*BE FALSE**************************)
1313 |                BEGIN(*1*)
1314 |                 IF SIMSAVB(.CBLOOPVAL.) THEN
1315 |                  BEGIN(*2*)
1316 |                   FOR VI:=1 TO MAXCELBLK DO
1317 |                    BEGIN
1318 |                     VSTOR:=UPDATYES(.VL,VI.);
1319 |                     UPDATYES(.VL,VI.):=SIMSAVB(.CBLOOPVAL.);
1320 |                     SIMSAVB(.CBLOOPVAL.):=VSTOR
1321 |                    END;
1322 |                  FOR VI:=1 TO MAXCELBLK DO
1323 |                   BEGIN
1324 |                    STOR:=UPADDGIV(.VL,VI.).IC;
1325 |                    UPADDGIV(.VL,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1326 |                    SIMSAVE(.CBLOOPVAL.):=STOR
1327 |                   END;
1328 |                  SIMSAVE(.CBLOOPVAL.):=VL;(*STORING FOR NEXT USE*)
1329 |                  SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1330 |                 END(*2*)
1331 |                ELSE(*IF SIMSAVB(.CBLOOPVAL.) IS FALSE*)
1332 |                 (*JUST STORE FOR THE NEXT VALUE*)
1333 |                   SIMSAVE(.CBLOOPVAL.):=VL;(*STORING FOR NEXT USE*)
1334 |                   SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1335 |            END(*1*)
1336 |         ELSE(*IF VL<= TO CBLOOP VAL*)
1337 |          BEGIN(*3*)
1338 |                IF SIMSAVB(.CBLOOPVAL.) THEN
1339 |                 BEGIN(*2*)
1340 |                  VGG:=SIMSAVE(.CBLOOPVAL.);
1341 |                  FOR VI:=1 TO MAXCELBLK DO
1342 |                   BEGIN
1343 |                    VSTOR:=UPDATYES(.VGG,VI.);
1344 |                    UPDATYES(.VGG,VI.):=SIMSAVB(.CBLOOPVAL.);
1345 |                    SIMSAVB(.CBLOOPVAL.):=VSTOR
1346 |                   END;
1347 |                  FOR VI:=1 TO MAXCELBLK DO
1348 |                   BEGIN
1349 |                    STOR:=UPADDGIV(.VGG,VI.).IC;
1350 |                    UPADDGIV(.VGG,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1351 |                    SIMSAVE(.CBLOOPVAL.):=STOR
1352 |                   END;
1353 |                   SIMSAVE(.CBLOOPVAL.):=0;
1354 |                   SIMSAVB(.CBLOOPVAL.):=FALSE
1355 |                 END;(*2*)
1356 |               VB:=TRUE;
1357 |               FOR VI:=1 TO MAXCELBLK DO
1358 |                BEGIN
1359 |                 VSTOR:=UPDATYES(.VL,VI.);
1360 |                 UPDATYES(.VL,VI.):=VB;
```

```
1362 |                END;
1363 |               VARX:=DISADDGIV.ADDRSLT2.IC;
1364 |                FOR VI:=1 TO MAXCELBLK DO
1365 |                 BEGIN
1366 |                  STOR:=UPADDGIV(.VL,VI.).IC;
1367 |                  UPADDGIV(.VL,VI.).IC:=VARX;
1368 |                  VARX:=STOR
1369 |                 END;
1370 |           END;(*3*)
1371 |         END;(*VL NOT EQUAL VK*)
1372 |   (*****************NEXT UPDATING THE CELL BLOCK*****************)
1373 |   (*************ACTUALLY WRITTING IN THE ARRY*WITH RESULT*****)
1374 |               VOPR:=DISADDGIV.ADDRSLT1.OPR;
1375 |               VARJ:=DISADDGIV.ADDRSLT1.CB;
1376 |               VARK:=DISADDGIV.ADDRSLT1.IC;
1377 |                IF (VOPR=1)THEN
1378 |                 CELLBLOCK(.VARJ,VARK.).OPRAND1:=DISADDGIV.RESULT;
1379 |                IF (VOPR=2)THEN
1380 |                 CELLBLOCK(.VARJ,VARK.).OPRAND2:=DISADDGIV.RESULT;
1381 |   (****GOING FOR WRITEUP OF SECOND ADDRESS*************)
1382 |   (****SAME RESULT WILL BE WRITTEN IN TWO PLACES********)
1383 |               VOPR:=DISADDGIV.ADDRSLT2.OPR;
1384 |               VARJ:=DISADDGIV.ADDRSLT2.CB;
1385 |               VARK:=DISADDGIV.ADDRSLT2.IC;
1386 |                IF (VOPR=1)THEN
1387 |                 CELLBLOCK(.VARJ,VARK.).OPRAND1:=DISADDGIV.RESULT;
1388 |                IF (VOPR=2)THEN
1389 |                 CELLBLOCK(.VARJ,VARK.).OPRAND2:=DISADDGIV.RESULT;
1390 |           END(*1A*)
1391 |         END;(*END BEGIN*)
1392 |       'C':(*SAME OPERATION AS 'A'*)
1393 |         BEGIN
1394 |          VK:=DISADDGIV.ADDRSLT1.CB;
1395 |          IF
1396 |           (((DISADDGIV.ADDRSLT1.CB)=0)AND((DISADDGIV.ADDRSLT1.IC)=0)AND
1397 |           ((DISADDGIV.ADDRSLT1.OPR)=0))THEN
1398 |   (***************CHECK THIS C B L O O P V A L*****************)
1399 |           BEGIN(*FALSE*)
1400 |             IF SIMSAVB(.CBLOOPVAL.) THEN
1401 |              BEGIN(*2*)
1402 |               VGG:=SIMSAVE(.CBLOOPVAL.);
1403 |               FOR VI:=1 TO MAXCELBLK DO
1404 |                BEGIN
1405 |                 VSTOR:=UPDATYES(.VGG,VI.);
1406 |                 UPDATYES(.VGG,VI.):=SIMSAVB(.CBLOOPVAL.);
1407 |                 SIMSAVB(.CBLOOPVAL.):=VSTOR
1408 |                END;
1409 |               FOR VI:=1 TO MAXCELBLK DO
1410 |                BEGIN
1411 |                 STOR:=UPADDGIV(.VGG,VI.).IC;
1412 |                 UPADDGIV(.VGG,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1413 |                 SIMSAVE(.CBLOOPVAL.):=STOR
1414 |                END;
1415 |                SIMSAVE(.CBLOOPVAL.):=0;
1416 |                SIMSAVB(.CBLOOPVAL.):=FALSE
1417 |              END;(*2*)
1418 |            VB:=FALSE;
1419 |            FOR VI:= 1 TO MAXCELBLK DO
1420 |             BEGIN(*1*)
1421 |              VSTOR:=UPDATYES(.CBLOOPVAL,VI.);
1422 |              UPDATYES(.CBLOOPVAL,VI.):=VB;
1423 |              VB:=VSTOR
1424 |             END;(*1*)
```

```
1427 |                  BEGIN
1428 |                   STOR:=UPADDGIV(.CBLOOPVAL,VI.).IC;
1429 |                   UPADDGIV(.CBLOOPVAL,VI.).IC:=VARX;
1430 |                   VARX:=STOR
1431 |                  END;
1432 |               END(*FALSE*)
1433 |              ELSE
1434 |               BEGIN(*1A*)
1435 |               (*GIVING THE ADDRESS VALUE TO THE FETHC UNIT,FOR*)
1436 |                    (*THE INSTRUCTION JUST UPDATED*)
1437 |                IF (VK > CBLOOPVAL) THEN(*BECAUSE FIRST TIME AROUND IT WOULD*)
1438 |                                        (*BE FALSE**************************)
1439 |                 BEGIN(*1*)
1440 |                  IF SIMSAVB(.CBLOOPVAL.) THEN
1441 |                   BEGIN(*2*)
1442 |                    FOR VI:=1 TO MAXCELBLK DO
1443 |                     BEGIN
1444 |                      VSTOR:=UPDATYES(.VK,VI.);
1445 |                      UPDATYES(.VK,VI.):=SIMSAVB(.CBLOOPVAL.);
1446 |                      SIMSAVB(.CBLOOPVAL.):=VSTOR
1447 |                     END;
1448 |                    FOR VI:=1 TO MAXCELBLK DO
1449 |                     BEGIN
1450 |                      STOR:=UPADDGIV(.VK,VI.).IC;
1451 |                      UPADDGIV(.VK,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1452 |                      SIMSAVE(.CBLOOPVAL.):=STOR
1453 |                     END;
1454 |                    SIMSAVE(.CBLOOPVAL.):=VK;(*STORING FOR NEXT USE*)
1455 |                    SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1456 |                   END(*2*)
1457 |                 ELSE(*IF SIMSAVB(.CBLOOPVAL.) IS FALSE*)
1458 |                  (*JUST STORE FOR THE NEXT VALUE*)
1459 |                    SIMSAVE(.CBLOOPVAL.):=VK;(*STORING FOR NEXT USE*)
1460 |                    SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1461 |            END(*1*)
1462 |          ELSE(*IF VK<= TO CBLOOP VAL*)
1463 |           BEGIN(*3*)
1464 |                 IF SIMSAVB(.CBLOOPVAL.) THEN
1465 |                  BEGIN(*2*)
1466 |                   VGG:=SIMSAVE(.CBLOOPVAL.);
1467 |                   FOR VI:=1 TO MAXCELBLK DO
1468 |                    BEGIN
1469 |                     VSTOR:=UPDATYES(.VGG,VI.);
1470 |                     UPDATYES(.VGG,VI.):=SIMSAVB(.CBLOOPVAL.);
1471 |                     SIMSAVB(.CBLOOPVAL.):=VSTOR
1472 |                    END;
1473 |                   FOR VI:=1 TO MAXCELBLK DO
1474 |                    BEGIN
1475 |                     STOR:=UPADDGIV(.VGG,VI.).IC;
1476 |                     UPADDGIV(.VGG,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1477 |                     SIMSAVE(.CBLOOPVAL.):=STOR
1478 |                    END;
1479 |                    SIMSAVE(.CBLOOPVAL.):=0;
1480 |                    SIMSAVB(.CBLOOPVAL.):=FALSE
1481 |                  END;(*2*)
1482 |                VB:=TRUE;
1483 |                FOR VI:=1 TO MAXCELBLK DO
1484 |                 BEGIN
1485 |                  VSTOR:=UPDATYES(.VK,VI.);
1486 |                  UPDATYES(.VK,VI.):=VB;
1487 |                  VB:=VSTOR
1488 |                 END;
1489 |               VARX:=DISADDGIV.ADDRSLT1.IC;
1490 |                FOR VI:=1 TO MAXCELBLK DO
```

```
1492 |                 STOR:=UPADDGIV(.VK,VI.).IC;
1493 |                 UPADDGIV(.VK,VI.).IC:=VARX;
1494 |                 VARX:=STOR
1495 |                END;
1496 |           END;(*3*)
1497 |  (****************NEXT UPDATING THE CELL BLOCK*****************)
1498 |  (*************ACTUALLY WRITTING IN THE ARRY*WITH RESULT******)
1499 |             VOPR:=DISADDGIV.ADDRSLT1.OPR;
1500 |             VARJ:=DISADDGIV.ADDRSLT1.CB;
1501 |             VARK:=DISADDGIV.ADDRSLT1.IC;
1502 |              IF (VOPR=1)THEN
1503 |               CELLBLOCK(.VARJ,VARK.).OPRAND1:=DISADDGIV.RESULT;
1504 |              IF (VOPR=2)THEN
1505 |               CELLBLOCK(.VARJ,VARK.).OPRAND2:=DISADDGIV.RESULT
1506 |            END(*END ELSE*)
1507 |          END;(*END BEGIN*)
1508 |        'D':(**SAME AS 'A' AND 'C' EXCEPT RESULT 2**)
1509 |          BEGIN
1510 |           VK:=DISADDGIV.ADDRSLT2.CB;
1511 |           IF
1512 |            (((DISADDGIV.ADDRSLT2.CB)=0)AND((DISADDGIV.ADDRSLT2.IC)=0)AND
1513 |            ((DISADDGIV.ADDRSLT2.OPR)=0))THEN
1514 |  (**************CHECK THIS C B L O O P V A L*****************)
1515 |            BEGIN(*FALSE*)
1516 |              IF SIMSAVB(.CBLOOPVAL.) THEN
1517 |               BEGIN(*2*)
1518 |                VGG:=SIMSAVE(.CBLOOPVAL.);
1519 |                FOR VI:=1 TO MAXCELBLK DO
1520 |                 BEGIN
1521 |                  VSTOR:=UPDATYES(.VGG,VI.);
1522 |                  UPDATYES(.VGG,VI.):=SIMSAVB(.CBLOOPVAL.);
1523 |                  SIMSAVB(.CBLOOPVAL.):=VSTOR
1524 |                 END;
1525 |                FOR VI:=1 TO MAXCELBLK DO
1526 |                 BEGIN
1527 |                  STOR:=UPADDGIV(.VGG,VI.).IC;
1528 |                  UPADDGIV(.VGG,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1529 |                  SIMSAVE(.CBLOOPVAL.):=STOR
1530 |                 END;
1531 |                 SIMSAVE(.CBLOOPVAL.):=0;
1532 |                 SIMSAVB(.CBLOOPVAL.):=FALSE
1533 |               END;(*2*)
1534 |             VB:=FALSE;
1535 |             FOR VI:= 1 TO MAXCELBLK DO
1536 |              BEGIN(*1*)
1537 |               VSTOR:=UPDATYES(.CBLOOPVAL,VI.);
1538 |               UPDATYES(.CBLOOPVAL,VI.):=VB;
1539 |               VB:=VSTOR
1540 |              END;(*1*)
1541 |               VARX:=0;
1542 |               FOR VI:=1 TO MAXCELBLK DO
1543 |                BEGIN
1544 |                 STOR:=UPADDGIV(.CBLOOPVAL,VI.).IC;
1545 |                 UPADDGIV(.CBLOOPVAL,VI.).IC:=VARX;
1546 |                 VARX:=STOR
1547 |                END;
1548 |            END(*FALSE*)
1549 |          ELSE
1550 |           BEGIN(*1A*)
1551 |          (*GIVING THE ADDRESS VALUE TO THE FETHC UNIT,FOR*)
1552 |                (*THE INSTRUCTION JUST UPDATED*)
1553 |            IF (VK > CBLOOPVAL) THEN(*BECAUSE FIRST TIME AROUND IT WOULD*)
1554 |                                     (*BE FALSE**************************)
1555 |             BEGIN(*1*)
```

```
1557 |                  BEGIN(*2*)
1558 |                   FOR VI:=1 TO MAXCELBLK DO
1559 |                    BEGIN
1560 |                     VSTOR:=UPDATYES(.VK,VI.);
1561 |                     UPDATYES(.VK,VI.):=SIMSAVB(.CBLOOPVAL.);
1562 |                     SIMSAVB(.CBLOOPVAL.):=VSTOR
1563 |                    END;
1564 |                   FOR VI:=1 TO MAXCELBLK DO
1565 |                    BEGIN
1566 |                     STOR:=UPADDGIV(.VK,VI.).IC;
1567 |                     UPADDGIV(.VK,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1568 |                     SIMSAVE(.CBLOOPVAL.):=STOR
1569 |                    END;
1570 |                   SIMSAVE(.CBLOOPVAL.):=VK;(*STORING FOR NEXT USE*)
1571 |                   SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1572 |                 END(*2*)
1573 |               ELSE(*IF SIMSAVB(.CBLOOPVAL.) IS FALSE*)
1574 |                (*JUST STORE FOR THE NEXT VALUE*)
1575 |                   SIMSAVE(.CBLOOPVAL.):=VK;(*STORING FOR NEXT USE*)
1576 |                   SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1577 |            END(*1*)
1578 |          ELSE(*IF VK<= TO CBLOOP VAL*)
1579 |           BEGIN(*3*)
1580 |                 IF SIMSAVB(.CBLOOPVAL.) THEN
1581 |                  BEGIN(*2*)
1582 |                   VGG:=SIMSAVE(.CBLOOPVAL.);
1583 |                   FOR VI:=1 TO MAXCELBLK DO
1584 |                    BEGIN
1585 |                     VSTOR:=UPDATYES(.VGG,VI.);
1586 |                     UPDATYES(.VGG,VI.):=SIMSAVB(.CBLOOPVAL.);
1587 |                     SIMSAVB(.CBLOOPVAL.):=VSTOR
1588 |                    END;
1589 |                   FOR VI:=1 TO MAXCELBLK DO
1590 |                    BEGIN
1591 |                     STOR:=UPADDGIV(.VGG,VI.).IC;
1592 |                     UPADDGIV(.VGG,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1593 |                     SIMSAVE(.CBLOOPVAL.):=STOR
1594 |                    END;
1595 |                   SIMSAVE(.CBLOOPVAL.):=0;
1596 |                   SIMSAVB(.CBLOOPVAL.):=FALSE
1597 |                  END;(*2*)
1598 |               VB:=TRUE;
1599 |               FOR VI:=1 TO MAXCELBLK DO
1600 |                BEGIN
1601 |                 VSTOR:=UPDATYES(.VK,VI.);
1602 |                 UPDATYES(.VK,VI.):=VB;
1603 |                 VB:=VSTOR
1604 |                END;
1605 |               VARX:=DISADDGIV.ADDRSLT2.IC;
1606 |                FOR VI:=1 TO MAXCELBLK DO
1607 |                 BEGIN
1608 |                  STOR:=UPADDGIV(.VK,VI.).IC;
1609 |                  UPADDGIV(.VK,VI.).IC:=VARX;
1610 |                  VARX:=STOR
1611 |                 END;
1612 |          END;(*3*)
1613 |   (***************NEXT UPDATING THE CELL BLOCK*****************)
1614 |   (*************ACTUALLY WRITTING IN THE ARRY*WITH RESULT*****)
1615 |               VOPR:=DISADDGIV.ADDRSLT2.OPR;
1616 |               VARJ:=DISADDGIV.ADDRSLT2.CB;
1617 |               VARK:=DISADDGIV.ADDRSLT2.IC;
1618 |                IF (VOPR=1)THEN
1619 |                 CELLBLOCK(.VARJ,VARK.).OPRAND1:=DISADDGIV.RESULT;
1620 |                IF (VOPR=2)THEN
```

```
1622 |            END;(*1A*)
1623 |          END;(*END BEGIN*)
1624 |      END(*CASE*)
1625 |  END;(*END PROCEDURE UPDATE*)
1626 |  (*************************************************************************)
1627 |  (*************************************************************************)
1628 |  (*************************************************************************)
1629 |  (**********************M A I N   P R O G R A M**************************)
1630 |  BEGIN(*NSIMARC*)
1631 |              (*INITILIZING ALL INSTRUCTIONS CELLS SUCH THAT OPR1*)
1632 |              (*OPR2 CONTAIN 99(IN HARDWARE IT WOULD BE FF) REPRESENTI*)
1633 |              (*ING BLANK RESULT=99,STAR=A,ADDRSLT1/2=0*)
1634 |              (*NOTE IN ACTUAL HARDWARE IMPLEMENTATION ,THIS WILL*)
1635 |              (*BE TAKEN CARE OF BY MASTER CONTROLLER USING*)
1636 |              (*CELL BLOCK MEM*)
1637 |    FOR I:=1 TO MAXCELBLK DO (*I FOR CELL*)
1638 |      BEGIN
1639 |       FOR J:=1 TO MAXIC DO(*J FOR IC*)
1640 |         BEGIN
1641 |          CELLBLOCK(.I,J.).OPCODE:=NOP;
1642 |          CELLBLOCK(.I,J.).OPRAND1:=99;
1643 |          CELLBLOCK(.I,J.).OPRAND2:=99;
1644 |          CELLBLOCK(.I,J.).RESULT:=99;
1645 |          CELLBLOCK(.I,J.).STAR:='A';
1646 |          CELLBLOCK(.I,J.).ADDRSLT1.CB:=0;
1647 |          CELLBLOCK(.I,J.).ADDRSLT1.IC:=0;
1648 |          CELLBLOCK(.I,J.).ADDRSLT1.OPR:=0;
1649 |          CELLBLOCK(.I,J.).ADDRSLT2.CB:=0;
1650 |          CELLBLOCK(.I,J.).ADDRSLT2.IC:=0;
1651 |          CELLBLOCK(.I,J.).ADDRSLT2.OPR:=0;
1652 |         END
1653 |      END;(*LOOP I,J FINISH*)
1654 |  (*************************************************************************)
1655 |  (**FLUSHING THE PROCESSING ELEMENT PIPELINE TO ZERO*)
1656 |  (**THIS IS THE SIMULATION OF HARDWARE INITILIZATION DONE BY **)
1657 |  (**MASTERCONTROLLER AT THE START OF DFEE****)
1658 |      FOR I:=1 TO MAXCELBLK DO
1659 |       BEGIN
1660 |         FOR J:=1 TO PIPELENGTH DO
1661 |           BEGIN
1662 |            PROCCALC(.I,J.).RESULT:=0;
1663 |            PROCADDRESS(.I,J.).ADDRSLT1.CB:=0;
1664 |            PROCADDRESS(.I,J.).ADDRSLT1.IC:=0;
1665 |            PROCADDRESS(.I,J.).ADDRSLT1.OPR:=0;
1666 |            PROCADDRESS(.I,J.).ADDRSLT2.CB:=0;
1667 |            PROCADDRESS(.I,J.).ADDRSLT2.IC:=0;
1668 |            PROCADDRESS(.I,J.).ADDRSLT2.OPR:=0;
1669 |            PROCADDRESS(.I,J.).STAR:='A'
1670 |           END
1671 |       END;(**LOOP**)
1672 |  (**FOR SIMULATION SOFTWARE WE NEED TO PUT THE UADDGIV TO ZERO**)
1673 |  (**ALSO INITILIZING UPDATYES AND OUTPTRDEY TO FALSE**)
1674 |      FOR I:=1 TO MAXCELBLK DO
1675 |        BEGIN
1676 |         FOR J:=1 TO MAXCELBLK DO
1677 |          BEGIN
1678 |            UPDATYES(.I,J.):=FALSE;
1679 |            OTPTREDY(.I,J.):=FALSE;
1680 |            UPADDGIV(.I,J.).CB:=0;
1681 |            UPADDGIV(.I,J.).IC:=0;
1682 |            UPADDGIV(.I,J.).OPR:=0
1683 |          END
1684 |        END;(**END LOOP**)
1685 |  (******INITILIZING COUNTERS*****)
```

```
1687 |          BEGIN
1688 |            COUNTICNUM(.I.):=1;
1689 |            FETCHREG(.I.):=0;
1690 |            SIMSAVE(.I.):=0;
1691 |            SIMSAVB(.I.):=FALSE
1692 |          END;
1693 |  (*****************************************************************)
1694 |  (*****************************************************************)
1695 |  (******ACTUAL INSTRUCTION CELL ASSIGNMENT ***********************)
1696 |  (***T H I S   I S   T H E   O U T P U T   O F   THE ***************)
1697 |  (***************O P T C O M P I L E R*************************)
1698 |  (*****************************************************************)
1699 |  (*****************************************************************)
1700 |  (******ACTUAL INSTRUCTION CELL ASSIGNMENT ***********************)
1701 |  (***T H I S   I S   T H E   O U T P U T   O F   THE ***************)
1702 |  (***************O P T C O M P I L E R*************************)
1703 |  (**CB1:IC:1**)
1704 |  CELLBLOCK(.1,1.).OPCODE:=ADD;
1705 |  CELLBLOCK(.1,1.).OPRAND1:=5;
1706 |  CELLBLOCK(.1,1.).OPRAND2:=4;
1707 |  CELLBLOCK(.1,1.).STAR:='A';
1708 |  CELLBLOCK(.1,1.).ADDRSLT1.CB:=1;
1709 |  CELLBLOCK(.1,1.).ADDRSLT1.IC:=2;
1710 |  CELLBLOCK(.1,1.).ADDRSLT1.OPR:=1;
1711 |  CELLBLOCK(.1,1.).ADDRSLT2.CB:=0;
1712 |  CELLBLOCK(.1,1.).ADDRSLT2.IC:=0;
1713 |  CELLBLOCK(.1,1.).ADDRSLT2.OPR:=0;
1714 |  (************************************)
1715 |  (****CB1:IC2***)
1716 |  CELLBLOCK(.1,2.).OPCODE:=MUL;
1717 |  CELLBLOCK(.1,2.).OPRAND1:=99;
1718 |  CELLBLOCK(.1,2.).OPRAND2:=99;
1719 |  CELLBLOCK(.1,2.).STAR:='A';
1720 |  CELLBLOCK(.1,2.).ADDRSLT1.CB:=2;
1721 |  CELLBLOCK(.1,2.).ADDRSLT1.IC:=2;
1722 |  CELLBLOCK(.1,2.).ADDRSLT1.OPR:=1;
1723 |  CELLBLOCK(.1,2.).ADDRSLT2.CB:=0;
1724 |  CELLBLOCK(.1,2.).ADDRSLT2.IC:=0;
1725 |  CELLBLOCK(.1,2.).ADDRSLT2.OPR:=0;
1726 |  (************************************)
1727 |  (***CB2:IC1******)
1728 |  CELLBLOCK(.2,1.).OPCODE:=SUB;
1729 |  CELLBLOCK(.2,1.).OPRAND1:=5;
1730 |  CELLBLOCK(.2,1.).OPRAND2:=4;
1731 |  CELLBLOCK(.2,1.).STAR:='A';
1732 |  CELLBLOCK(.2,1.).ADDRSLT1.CB:=1;
1733 |  CELLBLOCK(.2,1.).ADDRSLT1.IC:=2;
1734 |  CELLBLOCK(.2,1.).ADDRSLT1.OPR:=2;
1735 |  CELLBLOCK(.2,1.).ADDRSLT2.CB:=0;
1736 |  CELLBLOCK(.2,1.).ADDRSLT2.IC:=0;
1737 |  CELLBLOCK(.2,1.).ADDRSLT2.OPR:=0;
1738 |  (************************************)
1739 |  (***CB2:IC2******)
1740 |  CELLBLOCK(.2,2.).OPCODE:=FIN;
1741 |  CELLBLOCK(.2,2.).OPRAND1:=99;
1742 |  CELLBLOCK(.2,2.).OPRAND2:=99;
1743 |  CELLBLOCK(.2,2.).STAR:='A';
1744 |  CELLBLOCK(.2,2.).ADDRSLT1.CB:=0;
1745 |  CELLBLOCK(.2,2.).ADDRSLT1.IC:=0;
1746 |  CELLBLOCK(.2,2.).ADDRSLT1.OPR:=0;
1747 |  CELLBLOCK(.2,2.).ADDRSLT2.CB:=0;
1748 |  CELLBLOCK(.2,2.).ADDRSLT2.IC:=0;
1749 |  CELLBLOCK(.2,2.).ADDRSLT2.OPR:=0;
1750 |  (************************************)
```

```
1752 |  (**IN ACTUAL HARDWARE THIS WOULD BE DONE BY M/C WHICH AFTER**)
1753 |  (**RECEIVING REQUIRED OUTPUTS WILL STOP THE DFEE SYSTEM**)
1754 |     BEGIN(*MAIN PRG*)
1755 |      FLAGOUT:=FALSE;
1756 |      CLOCK:=1;
1757 |  (**STOP WILL BE USED TO STOP THE PROG.SIMULATION AS IF M/C STOPS IT*)
1758 |        WHILE NOT FLAGOUT DO
1759 |          (**IMPORTANT:NO STOPPING CRITERION SPECIFIED***JUST RUN***)
1760 |          BEGIN(*WHILE*)
1761 |           (**CLOCK TELLS US HOW MANUY TIMES THE MACHINE WILL RUN**)
1762 |           (**THROUGH ENCLOSING THE PARRALLEL LOOP**)
1763 |            FOR CBLOOPVAL:=1 TO MAXCELBLK DO
1764 |            BEGIN(*PARALLELISM LOOP*)
1765 |             FETCHUNIT;(*CALL FETCH UNIT*)
1766 |              WRITE('****FETCHUNIT OUTPUT***':8);
1767 |              WRITE('*FETCHUNIT NO':15,CBLOOPVAL:2);
1768 |              WRITE('*FETCHOPRND':13,ORD(FTHADGIV.OPCODE):2);
1769 |              WRITE('*FETCHEDOPR1':12,FTHADGIV.OPRAND1:2);
1770 |              WRITE('*FETCHEDOPR2':12,FTHADGIV.OPRAND2:2);
1771 |              WRITE('*FETCHED RES.BLK':16,FTHADGIV.RESULT:2);
1772 |              WRITE('*FETCHED*':9,FTHADGIV.STAR:3);
1773 |              WRITE('*FETCHED AD1.CB':15,FTHADGIV.ADDRSLT1.CB:2);
1774 |              WRITE('*FETCHED AD1.IC':15,FTHADGIV.ADDRSLT1.IC:2);
1775 |              WRITE('*FETCHED AD1.OPR':16,FTHADGIV.ADDRSLT1.OPR:2);
1776 |              WRITE('*FETCHED AD2.CB':15,FTHADGIV.ADDRSLT2.CB:2);
1777 |              WRITE('*FETCHED AD2.IC':15,FTHADGIV.ADDRSLT2.IC:2);
1778 |              WRITELN('*FETCHED AD2.OPR':16,FTHADGIV.ADDRSLT2.OPR:2);
1779 |              WRITE('***************PROCESSING ELEMENT*****':20);
1780 |  (****C H  E C K THE UPYES AND OTRDY SIG**MAY BE MISTAKE***)
1781 |             PROCELEM;(*CALL PROCELEMT*)
1782 |              WRITE('**RESULT PACKET**':12);
1783 |              WRITE('*RESULT TO DIS.NT':17,RESLTPAC.RESULT:2);
1784 |              WRITE('*ADDSTAR':8,RESLTPAC.STAR:4);
1785 |              WRITE('*ADD1CB':7,RESLTPAC.ADDRSLT1.CB:2);
1786 |              WRITE('*ADD1IC':7,RESLTPAC.ADDRSLT1.IC:2);
1787 |              WRITE('*ADD1OPR':8,RESLTPAC.ADDRSLT1.OPR:2);
1788 |              WRITE('*ADD2CB':7,RESLTPAC.ADDRSLT2.CB:2);
1789 |              WRITE('*ADD2IC':7,RESLTPAC.ADDRSLT2.IC:2);
1790 |              WRITELN('*ADD2OPR':8,RESLTPAC.ADDRSLT2.OPR:2);
1791 |   WRITE('************ D I S T.  N E T W O R K*******':20);
1792 |             DISTNET;(*CALLING DISTRIBUTION NETWORK*)
1793 |              WRITE('*DIS.RES.TRAN':13,DISADDGIV.RESULT:3);
1794 |              WRITE('*DIS.RES.STAR':13,DISADDGIV.STAR:3);
1795 |              WRITE('*DIS.ADD1.CB':12,DISADDGIV.ADDRSLT1.CB:2);
1796 |              WRITE('*DIS.ADD1.IC':12,DISADDGIV.ADDRSLT1.IC:2);
1797 |              WRITE('*DIS.ADD1.OPR':13,DISADDGIV.ADDRSLT1.OPR:2);
1798 |              WRITE('DIS.ADD2.CB':12,DISADDGIV.ADDRSLT2.CB:2);
1799 |              WRITE('DIS.ADD2.IC':12,DISADDGIV.ADDRSLT2.IC:2);
1800 |              WRITELN('DIS.ADD2.OPR':13,DISADDGIV.ADDRSLT2.OPR:2);
1801 |   WRITELN('COUNT OF CB':10,')':1,CBLOOPVAL:2,COUNTICNUM(.CBLOOPVAL.):2);
1802 |  WRITE('************U P D A T E  U N I T******':20);
1803 |             UPDATE;(*CALLING UPDATE UNIT*)
1804 |  WRITE('**UPDATEYES':11,CBLOOPVAL:1,')':1,UPDATYES(.CBLOOPVAL,MAXCELBLK.):8);
1805 |              WRITELN('***MEMDUMP FOLLOWS********':30);
1806 |               FOR II:=1 TO MAXCELBLK DO
1807 |                BEGIN
1808 |                 FOR JJ:=1 TO MAXIC DO
1809 |                  BEGIN
1810 |                   WRITE('CELLNO':6,II:2,'ICNO':4,JJ:2);
1811 |                   WRITE('*CELOPCD':8,ORD(CELLBLOCK(.II,JJ.).OPCODE):2
1812 |                   );
1813 |                   WRITE('*CELOPR1':8,CELLBLOCK(.II,JJ.).OPRAND1:2);
1814 |                   WRITE('*CELOPR2':8,CELLBLOCK(.II,JJ.).OPRAND2:2);
1815 |                   WRITE('*CELRES':7,CELLBLOCK(.II,JJ.).RESULT:2);
```

```
 1817  |                    WRITE('*CELAD1.CB':10,CELLBLOCK(.II,JJ.).ADDRSLT1.CB:2
 1818  |                     );
 1819  |                    WRITE('*CELAD1.IC':10,CELLBLOCK(.II,JJ.).ADDRSLT1.IC:2
 1820  |                     );
 1821  |                    WRITE('*CELAD1.OP':10,CELLBLOCK(.II,JJ.).ADDRSLT1.OPR:
 1822  |                     2);
 1823  |                    WRITE('*CELAD2.CB':10,CELLBLOCK(.II,JJ.).ADDRSLT2.CB:2
 1824  |                     );
 1825  |                    WRITE('*CELAD2.IC':10,CELLBLOCK(.II,JJ.).ADDRSLT2.IC:2
 1826  |                     );
 1827  |                    WRITELN('*CELAD2.OP':10,CELLBLOCK(.II,JJ.).ADDRSLT2.OPR:
 1828  |                     2);
 1829  |                    END
 1830  |                  END;(*PRINT LOOP*)
 1831  |                END;(**PARALLELISM LOOP**)
 1832  |             WRITELN('************************':10);
 1833  |             WRITELN('**CLOCK=':7,CLOCK:3);
 1834  |             WRITELN('************************':10);
 1835  |             CLOCK:=CLOCK+1
 1836  |           END;(*WHILE*)
 1837  |         END (*MAIN PRG*)
 1838  |       END.(*NSIMARC*)
 1839  |  %EOF
Execution begins...
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
****FETC  *FETCHUNIT NO 1  *FETCHOPRND 1*FETCHEDOPR1 5*FETCHEDOPR2 4*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 1*FETCHED AD1.IC 2
*FETCHED AD1.OPR 1*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
*****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************* D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 2
*************U P D A **UPDATEYES1)    FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 1*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 3*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 2*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 2*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 2*FETCHEDOPR1 5*FETCHEDOPR2 4*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 1*FETCHED AD1.IC 2
```

```
**********PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
********** D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 2
**********U P D A **UPDATEYES2)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 1*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 3*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 2*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 2*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK  1
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
****FETC  *FETCHUNIT NO 1  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
**********PROC**RESULT PAC*RESULT TO DIS.NT 9*ADDSTAR   A*ADD1CB 1*ADD1IC 2*ADD1OPR 1*ADD2CB 0*ADD2IC 0*ADD2OPR 0
********** D I S T*DIS.RES.TRAN  9*DIS.RES.STAR  A*DIS.ADD1.CB 1*DIS.ADD1.IC 2*DIS.ADD1.OPR 1 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 3
**********U P D A **UPDATEYES1)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 1*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 3*CELOPR1 9*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 2*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 2*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
```

```
**********PROC**RESULT PAC*RESULT TO DIS.NT 1*ADDSTAR   A*ADD1CB 1*ADD1IC 2*ADD1OPR 2*ADD2CB 0*ADD2IC 0*ADD2OPR 0
********** D I S T*DIS.RES.TRAN  1*DIS.RES.STAR  A*DIS.ADD1.CB 1*DIS.ADD1.IC 2*DIS.ADD1.OPR 2 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 3
**********U P D A **UPDATEYES2)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 1*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 3*CELOPR1 9*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 2*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 2*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK  2
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           2
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           2
++++D E B U G+++++
****FETC  *FETCHUNIT NO 1  *FETCHOPRND 3*FETCHEDOPR1 9*FETCHEDOPR2 1*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 2*FETCHED AD1.IC 2
*FETCHED AD1.OPR 1*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
**********PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
********** D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 3
**********U P D A **UPDATEYES1)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 1*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 3*CELOPR1 9*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 2*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 2*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
```

```
**********PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
********** D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 4
**********U P D A **UPDATEYES2)    FALSE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 1*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 3*CELOPR1 9*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 2*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 2*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK  3
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           2
++++D E B U G+++++
****FETC  *FETCHUNIT NO 1  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
**********PROC**RESULT PAC*RESULT TO DIS.NT 9*ADDSTAR   A*ADD1CB 2*ADD1IC 2*ADD1OPR 1*ADD2CB 0*ADD2IC 0*ADD2OPR 0
********** D I S T*DIS.RES.TRAN  9*DIS.RES.STAR  A*DIS.ADD1.CB 2*DIS.ADD1.IC 2*DIS.ADD1.OPR 1 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 4
**********U P D A **UPDATEYES1)    TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 1*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 3*CELOPR1 9*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 2*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 2*CELOPCD 9*CELOPR1 9*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
```

```
**********PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
********** D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 5
**********U P D A **UPDATEYES2)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 1*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 3*CELOPR1 9*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 2*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 2*CELOPCD 9*CELOPR1 9*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK  4
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS            2
++++D E B U G+++++
****FETC  *FETCHUNIT NO 1  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
**********PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
********** D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 5
**********U P D A **UPDATEYES1)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 1*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 3*CELOPR1 9*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 2*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 2*CELOPCD 9*CELOPR1 9*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS            2
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
```

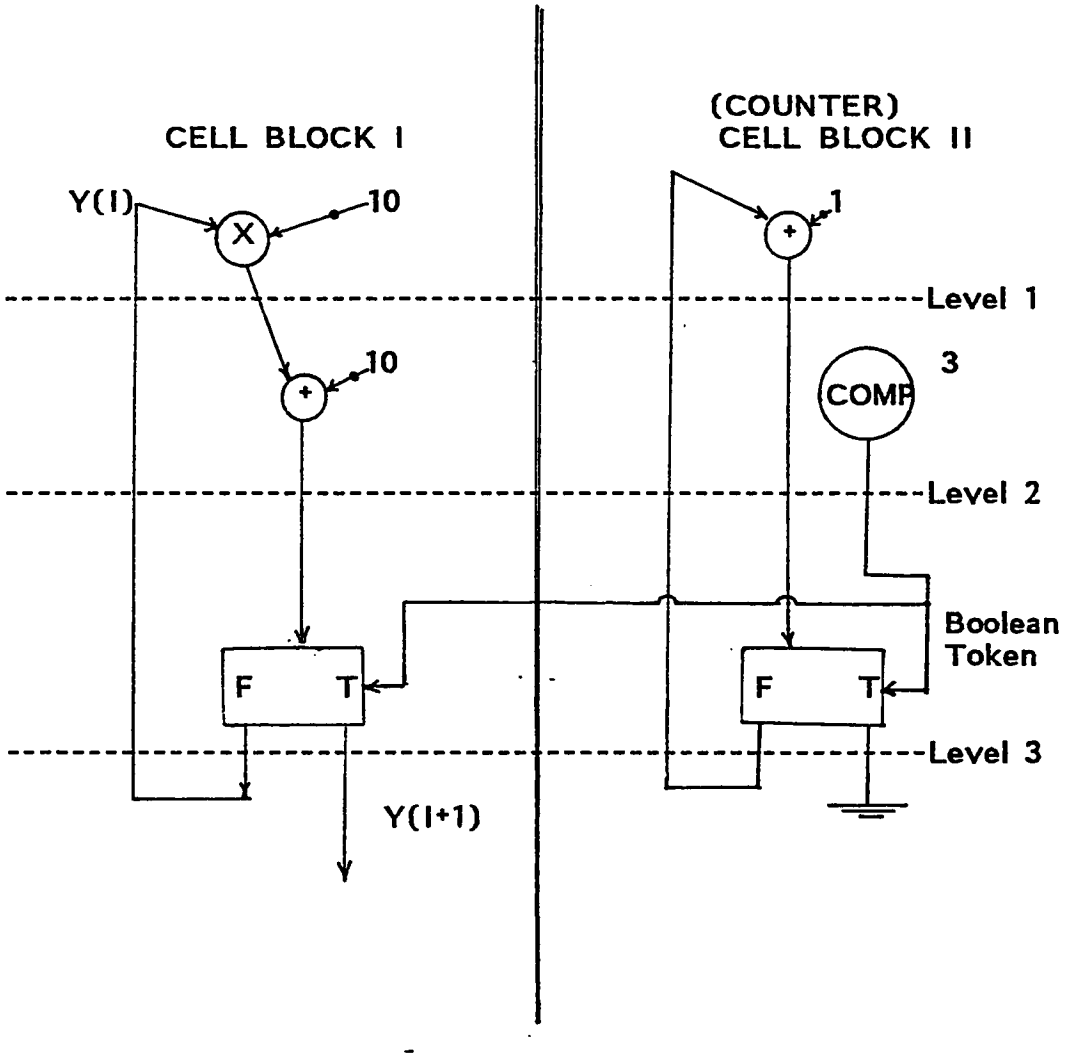
```
**********PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
********** D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 5
**********U P D A **UPDATEYES2)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 1*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 3*CELOPR1 9*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 2*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 2*CELOPCD 9*CELOPR1 9*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK  5
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
****FETC  *FETCHUNIT NO 1  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
**********PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
********** D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 5
**********U P D A **UPDATEYES1)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 1*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 3*CELOPR1 9*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 2*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 2*CELOPCD 9*CELOPR1 9*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS            2
++++D E B U G+++++
************************************************
```

```
  THE RESULT IS  9
**************************************************
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************* D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 5
*************U P D A **UPDATEYES2)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 1*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 3*CELOPR1 9*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 2*CELOPR1 5*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 2*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 3*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK  6
**********

...execution ends
File 'BSINGARC': 1839 lines; no diagnostics
106782 bytes of object code generated
4070 statements executed
164112 bytes of memory requested during compilation
5792 bytes returned before execution
10296 bytes requested during execution
```

APPENDIX B
Simulation Program and output listing for loop schema.

COMPUTATION:Y(I+1):=Y(I) x 10 + 10
For I=1 to 3;Y(1):=1;



```
  1 |  %$P 50000%
  2 |  %$X 50000%
  3 |  PROGRAM SIMARC(OUTPUT);
  4 |  (* COMMENTS REGARDING THE SIMULATION                                  *)
  5 |  (* 1A)THIS SIMULATION IS FOR SIGLE OUTPUT'FIN' EXPRESSION.*)
  6 |  (*    MODIFICATION REQUIRED FOR MULTIPLE OUTPUT INSTRUCTION*)
  7 |  (*    USING OUTPTREDY SIGNALARRY.
  8 |  (* 1) ALL THE VARIABLES ARE GLOBAL.I.E THERE NAME IS COMMUNICATED*)
  9 |  (*    DIRECTLY  TO ALL THE PROCEDURES REPRESENTING THE FUNCTIONALITY*)
 10 |  (*    OF THE SUB-UNITS.E.G FETCH-UNIT,UPDATE-UNIT,PRECES-ELEMENT,*)
 11 |  (*    DISTRIBUTION NETWORK.*)
 12 |  (* 2) THE PROCESSING ELEMENT IS REPRESENTED BY AN ARRAY OF*)
 13 |  (*    PIPELINE LENGTH 4, ALONG WITH THE ADDRESS PIPELINE          *)
 14 |  (*    OF THE SAME LENGTH. FURTHER, THE PR.EL DOES NOT TAKE CARE    *)
 15 |  (*    CARE OF SIGN IN THE SIMULATION.                              *)
 16 |  (* 3) TO REPRESENT THE CLOCK , A  OUTER LOOP IS USED*)
 17 |  (*    WHICH WILL BE EXITED WHEN THERE IS FIN OPCODE FOUND  *)
 18 |  (* 4) IMPORTANT: TO REPRESENT  THE PARALLEL EXECUTION OF CELL BLOCKS*)
 19 |  (*    ANOTHER DO LOOP RUNS THE SAME SET OF PROCEDURES NUMBER OF TIMES*)
 20 |  (*    WHICH ARE EQUAL TO THE CELL BLOCKS.THIS RESIDES INSIDE THE CLOCK*)
 21 |  (*    WHILE LOOP STATEMENT.*)
 22 |  (* 5) THERE IS A PROBLEM ENCOUNTERED WHILE RUNNING THE SIMULATION*)
 23 |  (*    OF PARALLEL EXECUTION OF CB. THE PROBLEM IS THAT CB(ITH) IS *)
 24 |  (*    UPDATING THE ITH +1 IN THE SAME CLOCK. THIS SHOULD BE DONE *)
 25 |  (*    IN THE NEXT CLOCK PHASE. FOR THIS PURPOSE, SIMSAVE AND SIMSAVB*)
 26 |  (*    HAVE BEEN DEFINED SO AS TO MAKE THE PARALLEL EXECUTION SIMULATION*)
 27 |  (* 6) NOTICE THE CONSTRAINT ON OPTIMIZATION COMPILER:*)
 28 |  (*    C O N S T R A I N T  ON THE OPTIMIZATION COMPILER*)
 29 |  (*    INSTRUCTION CELL INSTRUCTIONS ARE PLACED SUCH THAT THE S*)
 30 |  (*    UPADDGIV PIPELINE MAY NOT BE "OVERLOADED" AND LOOSE THE *)
 31 |  (*    ADDRESS OF THE IC JUST UPDATED.IE IF UPADDGIV PIPELINE FOR*)
 32 |  (*    CB(I)OS ALREADY FILLED ,THEN CB(J) WHERE J IS NOT EQUAT I*)
 33 |  (*    CANNOT PUT ANY ADDRESS IN CB(I) PIPELINE OF UPADDVIG WITHOUT*)
 34 |  (*    LOOSING ONE ADDRESS VALUE*)
 35 |  (*************************************************************************)
 36 |  CONST
 37 |   MAXIC=4;(*MAXIMUM NUMBER OF IC IN CELL BLOCK*)
 38 |   MAXCELBLK=2;(*MAX. NUMBER OF CELL BLOCKS IN ARCHITECTURE*)
 39 |   PIPELENGTH=2;(*PROCESSING ELEM. PIPELENGTH*)
 40 |  TYPE
 41 |   SUB1=1..MAXCELBLK;
 42 |   SUB2=1..MAXIC;
 43 |   SUB3=1..PIPELENGTH;
 44 |   NUMCELBLK=0..MAXCELBLK;(*0 JUST TO INCORPORATE NON VALID ADDRESS*)
 45 |                          (*FOR UPDATE UNIT                          *)
 46 |   NUMIC=0..MAXIC;(*ABOVE COMMENT APPLIES*)
 47 |   NUMOPR=0..2;(*SINCE THE NO OF OPR ARE ALWAYS EQUAL TO 2*)
 48 |   CODE=(NOP,ADD,SUB,MUL,DIVI,SQURT,SQAR,COMP,LOOPI,FIN);
 49 |   (*****COMMENT*************************************************)
 50 |   (*NOP SHOULD BE 0000 IN THE DESIGN BUT NOP IS NOT USED IN THE *)
 51 |   (*DESIGN BECAUSE EVERY IC HAS OPCODE OTHER THAN NOP WHEN FILLED*)
 52 |   (*BY OPTCOMPILER                                              *)
 53 |   (*************************************************************)
 54 |   STARVAL='A'..'D';
 55 |   (*************************************************************)
 56 |   (*****2-BIT REPRESENTATION,A=00,B=01,C=10,D=11*****************)
 57 |   (*****USED BY THE UPDATE UNIT*********************************)
 58 |   ADDSPEC=
 59 |       RECORD
 60 |          CB:NUMCELBLK;
```

```
63          END;
64      INSTRCELL=
65          RECORD
66              OPCODE:CODE;
67              OPRAND1,OPRAND2,RESULT:INTEGER;
68              STAR:STARVAL;
69              ADDRSLT1,ADDRSLT2:ADDSPEC
70          END;
71      (*****************************************************************)
72      (****EACH CELL BLOCK HAS FOUR INSTRUCTION CELLS. THIS WILL BE REPRESENTED*)
73      (****2-D CELL BLOCK ARRAY.SAMILARLY, NUMBER OF PE ARE EQUAL TO CB*)
74      (****AND EACH PE HAS 16 PIPELINE LENGTH**************************)
75      (****EACH CELL BLOCK FETCH UNIT HAS A COUNTER SO ARRAY************)
76      (*****************************************************************)
77      VAR
78          CELLBLOCK:ARRAY(.SUB1,SUB2.)OF INSTRCELL;
79          PROCCALC:ARRAY(.SUB1,SUB3.)OF INSTRCELL;
80      (*THIS ARRAY TAKES CARE OF CALCULATED RESULT*******************)
81      (**IN THE INITILIZATION OF THE MAIN PROGRAM, THIS ARRAY HAS TO *)
82      (**BE SET TO 0 IN ORDER TO SIMULATE THE 16CLOCK PIPLINE*********)
83          PROCADDRESS:ARRAY(.SUB1,SUB3.)OF INSTRCELL;
84      (**IN THE INITILIZATION OF THE MAIN PROGRAM, THIS ARRAY HAS TO *)
85      (**BE SET TO 0 IN ORDER TO SIMULATE THE 16CLOCK PIPLINE*********)
86      (**THIS ARRAY TAKES CARE OF SYNCHRONIZING THE RESULT WITH ITS****)
87      (***ADDRESS WHICH GOES PAST UNAFFECTED***************************)
88          COUNTICNUM:ARRAY(.SUB1.)OF INTEGER;
89      (*****************************************************************)
90      (*****************************************************************)
91      (************DEFINATION OF GLOBAL VARIABLES************************)
92      (***WHICH ARE CALLED BY THE SAME NAME IN THE PROCEDURE CALLS********)
93      (**********WHY?SEE COMMENT IN THE BEGINING************************)
94        II,JJ,STOP,I,J,K,L,RESULTOUT,CBLOOPVAL,CLOCK:INTEGER;
95      (*****RESULT OUT IS THE VALUE OF RESULT GIVEN TO THE MASTERCONTROLLER*)
96      (*****I,J,K,L ARE INDEX VARIABLES USED AT VARIOUS PLACES**************)
97      (***CBLOOPVAL IS USED FOR DETERMININMG THE PARALLEL EXECUTION******)
98      (***I,E CBLOOPVAL:=1 TO MAXCELBLK*********************************)
99        OTPTREDY,UPDATYES:ARRAY(.SUB1,SUB1.)OF BOOLEAN;
100       FLAGOUT:BOOLEAN;
101     (***FLAGOUT WILL BE MADE TRUE WHEN OPCODE "FIN" IS FOUND***)
102     (*****************************************************************)
103     (************NOTE:WE NEED AN AN ARRAY TO DISTUINGUISH BETWEEN ****)
104     (********THE TWO UPDATE YES SIGNALS AND OUTPUT READY SINCE WE ARE**)
105     (********USING LOOP FOR SIMULATING PARALLEL EXECUTION******)
106     (*********OTPTREDY  IS THE SIGNAL WHICH IS LOW BEFORE WE ENTER INTO**)
107     (****FETCH UNIT EXECUTION AND IT IS PUT HIGH BY FETCH UNIT IF "FIN"*)
108     (****IS FOUND BY THE FETCH UNIT,ELSE IT REMAINS LOW****************)
109     (*****************************************************************)
110     (****UPDATYES IS THE SIGNAL IF UDATE UNIT UPDATES THE IC AND THEN**)
111     (****SENDS THE CORROSPONDING ADDRESS TO THE FETCH UNIT*************)
112     (*****THIS ADDRESS CONTAINS THE CELL BLOCK NO. ,IC NO **)
113     (**HOWEVER, NOTE THAT NO OPRAND NUMBER IS GIVEN TO THE FETCHUNIT*)
114     (***NO NEED TO GIVE THE OPRAND NO TO THE FETCH UNIT**)
115     (******ALTHOUGH THE CEL BLOCK IS PRE DETERMINED BY THE DISTRIBUTION*)
116     (*NETWORK BUT SINCE DUE TO PARALLEL EXECUTION USING DO LOOP, WE NEED*)
117     (* SPECIFY ALL THE ADDRESS FIELD SO THE PROCEDUCRE CAN BE USED FOR*)
118     (*ALL CELL BLOCKS*)
119     (***UPADDGIVE-IS THE ADDRESSGIVEN BY UPDATE UNIT TO FETCH UNIT OF*)
120     (*THE ABOVE MENTIONED FORMAT*)
121     (***IT CONTAINS ONLY IC NUMBER THOUGH CB NUMBER IS CARRIED              *)
122     (***THROUGH IN SOFTWARE,THIS IS NOT DONE SO IN ACTUALL HARDWARE**)
123       UPADDGIV:ARRAY(.SUB1,SUB1.)OF ADDSPEC;
124     (*****************************************************************)
125       DISADDGIV:INSTRCELL;
```

```
127 |  (*NETWORK TO THE UPDATE UNIT AS TO WHICH CB THE RESULT IS COMING TO*)
128 |  (*AGAIN ,IN ACTUAL HARDWARE, THE CB NO. WILL BE USED UP BY THE *)
129 |  (*DISTRIBUTION NETWORK BUT SINCE IN SOFTWARE SIM. AN ARRAY IS CALLED*)
130 |  (*THEREFORE THE CB NO. WILL GO ALONG AS AN INDEX TO THE ARRAY CELBLK*)
131 |  (********THE FETCH UNIT FORMING OPERATION PACKET FOR THE PE GIVES**)
132 |  (********FTHADDGIV-TO THE PE SO IT CAN FORM THE RESULT PACKET******)
133 |  (*****SINCE THERE ARE TWO ADDRESSES ASSOCIATED WITH IT SO*******)
134 |  (**FTHADGIV-WE NEED A STAR AS WELL .UPADDGIVE NEED NOT GIVE THE**)
135 |  (**STAR SINCE ONLY ONE ADDRESS COMMUNICATED**)
136 |    FTHADGIV:INSTRCELL;
137 |  (****SO WE WILL SPECIFY THE WHOLE ADDRESS BLOCK AS**************)
138 |  (***FTHADGIV.STAR,FTHADGIV.ADDRSLT1.CB,FTHADGIV.ADDRSLT1.IC,**)
139 |  (****FTHADGIV.ADDRSLT1.OPR, AND SAMILARLY FOR ADDRSLT2.*******)
140 |  (*************************************************************)
141 |  (*****THE RESULT PACKET OUT OF THE PRECESSING ELEMENT CONTAINS**)
142 |  (*************************************************************)
143 |  (*RESULT *STAR*ADDRESULT1*ADDRESULT2**)
144 |  (*************************************************************)
145 |  (****THIS GOES TO THE DISTRIBUTION NETWORK AND THEN TO THE UPDATE*)
146 |  (*****UNIT.AGAIN ONE THING TO NOTE IS UPDATE UNIT MAY UPDATE******)
147 |  (****MORE INSTRUCTIONS DEPENDING ON STAR (**CONFLICT SHOULD BE*****)
148 |  (****AVOIDED BY THE OPTCOMPILER.I.E WITH THE UPDATE YES THERE SHOULD*)
149 |  (***BE NO TWO INSTRUCTIONS ENABLED IN THE SAME CELL BLOCK AT THE SAME*)
150 |  (****TIME. BUT THE UPDATE SIGNAL SENDS ONLY ONE ADDRESS TO THE FETCH*)
151 |  (*UNIT DEPENDING ON THE STAR .SEE THE FLOWCHART*)
152 |  (*************************************************************)
153 |    RESLTPAC:INSTRCELL;
154 |  (**RESLTPAC IS THE FORMAT OF THE RESULT PACKET AFTER THE RESULT IS **)
155 |  (*OUT OF THE PROCESSING ELEMENT ANDIS SYN. WITH THE ADDRESS.*********)
156 |    FETCHREG:ARRAY(.SUB1.)OF INTEGER;
157 |  (*THIS REGISTER WILL RESIDE IN EACH FETCH UNIT AND WILL MAKE SURE**)
158 |  (**THAT SAME IC IS NOT FETCHED "MORE THAN ONCE IN A ROW"*********)
159 |    SIMSAVE:ARRAY(.SUB1.)OF INTEGER;
160 |    SIMSAVB:ARRAY(.SUB1.)OF BOOLEAN;
161 |  (**  ONLY USED IN ORDER TO SOLVE THE PROBLEM OF PARALLEL EXECUTION*(
162 |  (**  SO THAT CBI DOES NOT EXECUTE CB(I+1) IN THE SAME CLOCK**)
163 |  (******I M P O R T A N T*************************************)
164 |  (*** NOT TO BE REPRESENTED IN HARDWARE*******)
165 |  (********FUNCTIONAL DEFINATION OF THE SUB-UNITS REPRESENTED BY THE **)
166 |  (**REMEMBER , COUNTICNUM=1,OTPTREDY=FALSE,UPDATYES=FALSE INITIALLY***)
167 |  (***BEFORE THE EXECUTION OF THE PROCEDURE AND AFTER THE EXECUTION OF*)
168 |  (***THE PROCEDURE**THIS WOULD BE DONE IN THE MAIN PROGRAM ***********)
169 |  (***PROCEDURES STARTS HERE*************************************)
170 |  (***REMEMBER:ALL VAR GLOBALI.E DIRECTLY AFFECT THE MAIN PROGRAM**)
171 |  PROCEDURE FETCHUNIT;
172 |   VAR
173 |      IL:INTEGER;
174 |  BEGIN(*111*)
175 |   FOR IL:=1 TO MAXCELBLK DO
176 |    BEGIN
177 |   WRITELN('++++D E B U G+++++');
178 |   WRITELN('FETCH UNIT PIPLINE UP YES',UPDATYES(.CBLOOPVAL,IL.));
179 |   WRITELN('FET UNIT PIPLINE UPADDRESS',UPADDGIV(.CBLOOPVAL,IL.).IC);
180 |   WRITELN('++++D E B U G+++++')
181 |    END;
182 |   BEGIN(**PROCEDUREUPDATE*)
183 |   J:=CBLOOPVAL;(*J REMAINS SAME FOR FETCH UNIT*)
184 |   IF UPDATYES(.CBLOOPVAL,MAXCELBLK.) THEN
185 |     BEGIN(*TRUE UPDATEYES*)
186 |       K:=UPADDGIV(.CBLOOPVAL,MAXCELBLK.).IC;
187 |       IF(CELLBLOCK(.J,K.).OPCODE=FIN) THEN
188 |         BEGIN(*FIN *)
189 |          OTPTREDY(.CBLOOPVAL,MAXCELBLK.):=TRUE;
190 |          FLAGOUT:=OTPTREDY(.CBLOOPVAL,MAXCELBLK.);
```

```
192 |            WRITELN('FIN OPCODE FOUND FOR CELL ':18,CBLOOPVAL:3);
193 |            RESULTOUT:=CELLBLOCK(.J,K.).OPRAND1;
194 |            WRITELN('THE RESULT IS':15,RESULTOUT:3);
195 |            WRITELN('************************************************');
196 |            CELLBLOCK(.J,K.).OPCODE:=NOP;
197 |            CELLBLOCK(.J,K.).OPRAND1:=99;
198 |            CELLBLOCK(.J,K.).OPRAND2:=99;
199 |            CELLBLOCK(.J,K.).RESULT:=99;
200 |            CELLBLOCK(.J,K.).STAR:='A';
201 |            CELLBLOCK(.J,K.).ADDRSLT1.CB:=0;
202 |            CELLBLOCK(.J,K.).ADDRSLT1.IC:=0;
203 |            CELLBLOCK(.J,K.).ADDRSLT1.OPR:=0;
204 |            CELLBLOCK(.J,K.).ADDRSLT2.CB:=0;
205 |            CELLBLOCK(.J,K.).ADDRSLT2.IC:=0;
206 |            CELLBLOCK(.J,K.).ADDRSLT2.OPR:=0
207 |          END(*FIN*)
208 |         ELSE
209 |          BEGIN(*NOT EQ TO FIN*)
210 |          IF(((CELLBLOCK(.J,K.).OPCODE)=SQURT)OR ((CELLBLOCK(.J,K.).OPCODE
211 |             =SQAR)))THEN
212 |    (**NOTICE THIS IS ONLY TESTING FOR OPRAND 1 TO BE FULL FOR **)
213 |    (*****UNI-OPERATION CODE***R-FF HAS TO HAVE PROVISION FOR THIS******)
214 |              BEGIN(*AA*)
215 |               IF((CELLBLOCK(.J,K.).OPRAND1 <> 99))THEN
216 |                 BEGIN(*A1*)
217 |                  IF(K<>FETCHREG(.J.)) THEN
218 |                   BEGIN(*A2*)
219 |                    FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
220 |                    FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
221 |                    FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
222 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
223 |                    FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
224 |                    FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
225 |                    FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
226 |                    FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
227 |                    FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
228 |                    FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
229 |                    FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
230 |                    FETCHREG(.J.):=K
231 |                  END(*A2*)
232 |                   ELSE
233 |        BEGIN(*A2 FALSE*)
234 |         K:=COUNTICNUM(.J.);
235 |          IF (K<= MAXIC)THEN
236 |           BEGIN (*BB1*)
237 |            COUNTICNUM(.J.):=COUNTICNUM(.J.)+1;
238 |          IF(((CELLBLOCK(.J,K.).OPCODE)=SQURT)OR ((CELLBLOCK(.J,K.).OPCODE
239 |             =SQAR)))THEN
240 |              BEGIN(*BB2*)
241 |               IF((CELLBLOCK(.J,K.).OPRAND1 <> 99))THEN
242 |                BEGIN(*BB3*)
243 |                 IF(K<>FETCHREG(.J.))THEN
244 |                  BEGIN(*BB4*)
245 |                   FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
246 |                   FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
247 |                   FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
248 |                   FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
249 |                   FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
250 |                   FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
251 |                   FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
252 |                   FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
253 |                   FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
254 |                   FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
255 |                   FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
```

```
257 |                  END(*BB4*)
258 |                 ELSE
259 |                  BEGIN(*BB4 FLASE*)
260 |                   FTHADGIV.OPCODE:=NOP;
261 |                   FTHADGIV.OPRAND1:=0;
262 |                   FTHADGIV.OPRAND2:=0;
263 |                   FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
264 |                   FTHADGIV.STAR:='A';
265 |                   FTHADGIV.ADDRSLT1.CB:=0;
266 |                   FTHADGIV.ADDRSLT1.IC:=0;
267 |                   FTHADGIV.ADDRSLT1.OPR:=0;
268 |                   FTHADGIV.ADDRSLT2.CB:=0;
269 |                   FTHADGIV.ADDRSLT2.IC:=0;
270 |                   FTHADGIV.ADDRSLT2.OPR:=0
271 |                  END;(*BB4 FALSE*)
272 |                END(*BB3*)
273 |               ELSE
274 |                BEGIN(*BB3 FALSE*)
275 |                   FTHADGIV.OPCODE:=NOP;
276 |                   FTHADGIV.OPRAND1:=0;
277 |                   FTHADGIV.OPRAND2:=0;
278 |                   FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
279 |                   FTHADGIV.STAR:='A';
280 |                   FTHADGIV.ADDRSLT1.CB:=0;
281 |                   FTHADGIV.ADDRSLT1.IC:=0;
282 |                   FTHADGIV.ADDRSLT1.OPR:=0;
283 |                   FTHADGIV.ADDRSLT2.CB:=0;
284 |                   FTHADGIV.ADDRSLT2.IC:=0;
285 |                   FTHADGIV.ADDRSLT2.OPR:=0
286 |                 END;(*BB3 FALSE*)
287 |              END(*BB2*)
288 |             ELSE(*BB2 FALSE*)
289 |              BEGIN(*BB2F1*)
290 |          IF((CELLBLOCK(.J,K.).OPRAND1<>99)AND(CELLBLOCK(.J,K.).OPRAND2<>99)) THEN
291 |              BEGIN(**BB2 F2*)
292 |               IF(K<>FETCHREG(.J.))THEN
293 |                BEGIN(**BB2 F3*)
294 |                   FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
295 |                   FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
296 |                   FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
297 |                   FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
298 |                   FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
299 |                   FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
300 |                   FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
301 |                   FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
302 |                   FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
303 |                   FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
304 |                   FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
305 |                   FETCHREG(.J.):=K
306 |                 END(*BB2 F3*)
307 |               ELSE(*BB2 F3 FLASE*)
308 |                BEGIN(*BB2 F3 FALSE*)
309 |                   FTHADGIV.OPCODE:=NOP;
310 |                   FTHADGIV.OPRAND1:=0;
311 |                   FTHADGIV.OPRAND2:=0;
312 |                   FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
313 |                   FTHADGIV.STAR:='A';
314 |                   FTHADGIV.ADDRSLT1.CB:=0;
315 |                   FTHADGIV.ADDRSLT1.IC:=0;
316 |                   FTHADGIV.ADDRSLT1.OPR:=0;
317 |                   FTHADGIV.ADDRSLT2.CB:=0;
318 |                   FTHADGIV.ADDRSLT2.IC:=0;
319 |                   FTHADGIV.ADDRSLT2.OPR:=0
320 |                END;(*BB2 F3 FALSE*)
```

```
322 |            ELSE(*BB2 F2 FALSE*)
323 |             BEGIN(*BB2 F2 FALSE*)
324 |                    FTHADGIV.OPCODE:=NOP;
325 |                    FTHADGIV.OPRAND1:=0;
326 |                    FTHADGIV.OPRAND2:=0;
327 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
328 |                    FTHADGIV.STAR:='A';
329 |                    FTHADGIV.ADDRSLT1.CB:=0;
330 |                    FTHADGIV.ADDRSLT1.IC:=0;
331 |                    FTHADGIV.ADDRSLT1.OPR:=0;
332 |                    FTHADGIV.ADDRSLT2.CB:=0;
333 |                    FTHADGIV.ADDRSLT2.IC:=0;
334 |                    FTHADGIV.ADDRSLT2.OPR:=0
335 |             END;(*BB2 F2 FALSE*)
336 |           END;(*BB2 F1*)
337 |          END(*BB1*)
338 |          ELSE(*BB1 FALSE*)
339 |           BEGIN(*BB1 FALSE*)
340 |                    FTHADGIV.OPCODE:=NOP;
341 |                    FTHADGIV.OPRAND1:=0;
342 |                    FTHADGIV.OPRAND2:=0;
343 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
344 |                    FTHADGIV.STAR:='A';
345 |                    FTHADGIV.ADDRSLT1.CB:=0;
346 |                    FTHADGIV.ADDRSLT1.IC:=0;
347 |                    FTHADGIV.ADDRSLT1.OPR:=0;
348 |                    FTHADGIV.ADDRSLT2.CB:=0;
349 |                    FTHADGIV.ADDRSLT2.IC:=0;
350 |                    FTHADGIV.ADDRSLT2.OPR:=0
351 |           END;(*BB1 FALSE*)
352 |        END;(*A2 FALSE*)
353 |                 END(*A1*)
354 |               ELSE
355 |         BEGIN (*A1 FALSE*)
356 |          K:=COUNTICNUM(.J.);
357 |           IF (K<= MAXIC)THEN
358 |            BEGIN (*BB1*)
359 |             COUNTICNUM(.J.):=COUNTICNUM(.J.)+1;
360 |           IF(((CELLBLOCK(.J,K.).OPCODE)=SQURT)OR ((CELLBLOCK(.J,K.).OPCODE
361 |              =SQAR)))THEN
362 |               BEGIN(*BB2*)
363 |                IF((CELLBLOCK(.J,K.).OPRAND1 <> 99))THEN
364 |                 BEGIN(*BB3*)
365 |                  IF(K<>FETCHREG(.J.))THEN
366 |                   BEGIN(*BB4*)
367 |                    FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
368 |                    FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
369 |                    FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
370 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
371 |                    FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
372 |                    FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
373 |                    FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
374 |                    FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
375 |                    FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
376 |                    FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
377 |                    FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
378 |                    FETCHREG(.J.):=K
379 |                   END(*BB4*)
380 |                  ELSE
381 |                   BEGIN(*BB4 FLASE*)
382 |                    FTHADGIV.OPCODE:=NOP;
383 |                    FTHADGIV.OPRAND1:=0;
384 |                    FTHADGIV.OPRAND2:=0;
385 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
```

```
387 |                    FTHADGIV.ADDRSLT1.CB:=0;
388 |                    FTHADGIV.ADDRSLT1.IC:=0;
389 |                    FTHADGIV.ADDRSLT1.OPR:=0;
390 |                    FTHADGIV.ADDRSLT2.CB:=0;
391 |                    FTHADGIV.ADDRSLT2.IC:=0;
392 |                    FTHADGIV.ADDRSLT2.OPR:=0
393 |                  END;(*BB4 FALSE*)
394 |                END(*BB3*)
395 |               ELSE
396 |                BEGIN(*BB3 FALSE*)
397 |                    FTHADGIV.OPCODE:=NOP;
398 |                    FTHADGIV.OPRAND1:=0;
399 |                    FTHADGIV.OPRAND2:=0;
400 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
401 |                    FTHADGIV.STAR:='A';
402 |                    FTHADGIV.ADDRSLT1.CB:=0;
403 |                    FTHADGIV.ADDRSLT1.IC:=0;
404 |                    FTHADGIV.ADDRSLT1.OPR:=0;
405 |                    FTHADGIV.ADDRSLT2.CB:=0;
406 |                    FTHADGIV.ADDRSLT2.IC:=0;
407 |                    FTHADGIV.ADDRSLT2.OPR:=0
408 |                END;(*BB3 FALSE*)
409 |              END(*BB2*)
410 |             ELSE(*BB2 FALSE*)
411 |              BEGIN(*BB2F1*)
412 |          IF((CELLBLOCK(.J,K.).OPRAND1<>99)AND(CELLBLOCK(.J,K.).OPRAND2<>99)) THEN
413 |              BEGIN(**BB2 F2*)
414 |               IF(K<>FETCHREG(.J.))THEN
415 |                BEGIN(**BB2 F3*)
416 |                    FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
417 |                    FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
418 |                    FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
419 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
420 |                    FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
421 |                    FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
422 |                    FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
423 |                    FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
424 |                    FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
425 |                    FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
426 |                    FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
427 |                    FETCHREG(.J.):=K
428 |                END(*BB2 F3*)
429 |               ELSE(*BB2 F3 FLASE*)
430 |                BEGIN(*BB2 F3 FALSE*)
431 |                    FTHADGIV.OPCODE:=NOP;
432 |                    FTHADGIV.OPRAND1:=0;
433 |                    FTHADGIV.OPRAND2:=0;
434 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
435 |                    FTHADGIV.STAR:='A';
436 |                    FTHADGIV.ADDRSLT1.CB:=0;
437 |                    FTHADGIV.ADDRSLT1.IC:=0;
438 |                    FTHADGIV.ADDRSLT1.OPR:=0;
439 |                    FTHADGIV.ADDRSLT2.CB:=0;
440 |                    FTHADGIV.ADDRSLT2.IC:=0;
441 |                    FTHADGIV.ADDRSLT2.OPR:=0
442 |                END;(*BB2 F3 FALSE*)
443 |             END(*BB2 F2*)
444 |            ELSE(*BB2 F2 FALSE*)
445 |             BEGIN(*BB2 F2 FALSE*)
446 |                    FTHADGIV.OPCODE:=NOP;
447 |                    FTHADGIV.OPRAND1:=0;
448 |                    FTHADGIV.OPRAND2:=0;
449 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
450 |                    FTHADGIV.STAR:='A';
```

```
452 |                   FTHADGIV.ADDRSLT1.IC:=0;
453 |                   FTHADGIV.ADDRSLT1.OPR:=0;
454 |                   FTHADGIV.ADDRSLT2.CB:=0;
455 |                   FTHADGIV.ADDRSLT2.IC:=0;
456 |                   FTHADGIV.ADDRSLT2.OPR:=0
457 |            END;(*BB2 F2 FALSE*)
458 |           END;(*BB2 F1*)
459 |          END(*BB1*)
460 |          ELSE(*BB1 FALSE*)
461 |           BEGIN(*BB1 FALSE*)
462 |                   FTHADGIV.OPCODE:=NOP;
463 |                   FTHADGIV.OPRAND1:=0;
464 |                   FTHADGIV.OPRAND2:=0;
465 |                   FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
466 |                   FTHADGIV.STAR:='A';
467 |                   FTHADGIV.ADDRSLT1.CB:=0;
468 |                   FTHADGIV.ADDRSLT1.IC:=0;
469 |                   FTHADGIV.ADDRSLT1.OPR:=0;
470 |                   FTHADGIV.ADDRSLT2.CB:=0;
471 |                   FTHADGIV.ADDRSLT2.IC:=0;
472 |                   FTHADGIV.ADDRSLT2.OPR:=0
473 |           END;(*BB1 FALSE*)
474 |          END;(*A1 FALSE*)
475 |             END(*AA*)
476 |          ELSE
477 |             BEGIN (*AA FALSE*)
478 |          IF((CELLBLOCK(.J,K.).OPRAND1<>99)AND(CELLBLOCK(.J,K.).OPRAND2<>99)) THEN
479 |          BEGIN(*AA1*)
480 |           IF(K<>FETCHREG(.J.))THEN
481 |            BEGIN(*AA2*)
482 |                   FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
483 |                   FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
484 |                   FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
485 |                   FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
486 |                   FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
487 |                   FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
488 |                   FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
489 |                   FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
490 |                   FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
491 |                   FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
492 |                   FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
493 |            FETCHREG(.J.):=K
494 |            END(**AA2*)
495 |           ELSE
496 |         BEGIN(*AA2 FALSE*)
497 |          K:=COUNTICNUM(.J.);
498 |           IF (K<= MAXIC)THEN
499 |            BEGIN (*BB1*)
500 |             COUNTICNUM(.J.):=COUNTICNUM(.J.)+1;
501 |           IF(((CELLBLOCK(.J,K.).OPCODE)=SQURT)OR ((CELLBLOCK(.J,K.).OPCODE
502 |              =SQAR)))THEN
503 |               BEGIN(*BB2*)
504 |                IF((CELLBLOCK(.J,K.).OPRAND1 <> 99))THEN
505 |                BEGIN(*BB3*)
506 |                 IF(K<>FETCHREG(.J.))THEN
507 |                  BEGIN(*BB4*)
508 |                   FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
509 |                   FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
510 |                   FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
511 |                   FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
512 |                   FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
513 |                   FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
514 |                   FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
515 |                   FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
```

```
517 |                    FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
518 |                    FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
519 |                    FETCHREG(.J.):=K
520 |                   END(*BB4*)
521 |                  ELSE
522 |                   BEGIN(*BB4 FLASE*)
523 |                    FTHADGIV.OPCODE:=NOP;
524 |                    FTHADGIV.OPRAND1:=0;
525 |                    FTHADGIV.OPRAND2:=0;
526 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
527 |                    FTHADGIV.STAR:='A';
528 |                    FTHADGIV.ADDRSLT1.CB:=0;
529 |                    FTHADGIV.ADDRSLT1.IC:=0;
530 |                    FTHADGIV.ADDRSLT1.OPR:=0;
531 |                    FTHADGIV.ADDRSLT2.CB:=0;
532 |                    FTHADGIV.ADDRSLT2.IC:=0;
533 |                    FTHADGIV.ADDRSLT2.OPR:=0
534 |                   END;(*BB4 FALSE*)
535 |                 END(*BB3*)
536 |                ELSE
537 |                 BEGIN(*BB3 FALSE*)
538 |                    FTHADGIV.OPCODE:=NOP;
539 |                    FTHADGIV.OPRAND1:=0;
540 |                    FTHADGIV.OPRAND2:=0;
541 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
542 |                    FTHADGIV.STAR:='A';
543 |                    FTHADGIV.ADDRSLT1.CB:=0;
544 |                    FTHADGIV.ADDRSLT1.IC:=0;
545 |                    FTHADGIV.ADDRSLT1.OPR:=0;
546 |                    FTHADGIV.ADDRSLT2.CB:=0;
547 |                    FTHADGIV.ADDRSLT2.IC:=0;
548 |                    FTHADGIV.ADDRSLT2.OPR:=0
549 |                 END;(*BB3 FALSE*)
550 |               END(*BB2*)
551 |              ELSE(*BB2 FALSE*)
552 |               BEGIN(*BB2F1*)
553 |          IF((CELLBLOCK(.J,K.).OPRAND1<>99)AND(CELLBLOCK(.J,K.).OPRAND2<>99)) THEN
554 |               BEGIN(**BB2 F2*)
555 |                IF(K<>FETCHREG(.J.))THEN
556 |                 BEGIN(**BB2 F3*)
557 |                    FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
558 |                    FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
559 |                    FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
560 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
561 |                    FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
562 |                    FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
563 |                    FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
564 |                    FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
565 |                    FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
566 |                    FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
567 |                    FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
568 |                    FETCHREG(.J.):=K
569 |                 END(*BB2 F3*)
570 |               ELSE(*BB2 F3 FLASE*)
571 |                BEGIN(*BB2 F3 FALSE*)
572 |                    FTHADGIV.OPCODE:=NOP;
573 |                    FTHADGIV.OPRAND1:=0;
574 |                    FTHADGIV.OPRAND2:=0;
575 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
576 |                    FTHADGIV.STAR:='A';
577 |                    FTHADGIV.ADDRSLT1.CB:=0;
578 |                    FTHADGIV.ADDRSLT1.IC:=0;
579 |                    FTHADGIV.ADDRSLT1.OPR:=0;
580 |                    FTHADGIV.ADDRSLT2.CB:=0;
```

```
583 |                 END;(*BB2 F3 FALSE*)
584 |               END(*BB2 F2*)
585 |              ELSE(*BB2 F2 FALSE*)
586 |               BEGIN(*BB2 F2 FALSE*)
587 |                       FTHADGIV.OPCODE:=NOP;
588 |                       FTHADGIV.OPRAND1:=0;
589 |                       FTHADGIV.OPRAND2:=0;
590 |                       FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
591 |                       FTHADGIV.STAR:='A';
592 |                       FTHADGIV.ADDRSLT1.CB:=0;
593 |                       FTHADGIV.ADDRSLT1.IC:=0;
594 |                       FTHADGIV.ADDRSLT1.OPR:=0;
595 |                       FTHADGIV.ADDRSLT2.CB:=0;
596 |                       FTHADGIV.ADDRSLT2.IC:=0;
597 |                       FTHADGIV.ADDRSLT2.OPR:=0
598 |               END;(*BB2 F2 FALSE*)
599 |              END;(*BB2 F1*)
600 |             END(*BB1*)
601 |             ELSE(*BB1 FALSE*)
602 |              BEGIN(*BB1 FALSE*)
603 |                       FTHADGIV.OPCODE:=NOP;
604 |                       FTHADGIV.OPRAND1:=0;
605 |                       FTHADGIV.OPRAND2:=0;
606 |                       FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
607 |                       FTHADGIV.STAR:='A';
608 |                       FTHADGIV.ADDRSLT1.CB:=0;
609 |                       FTHADGIV.ADDRSLT1.IC:=0;
610 |                       FTHADGIV.ADDRSLT1.OPR:=0;
611 |                       FTHADGIV.ADDRSLT2.CB:=0;
612 |                       FTHADGIV.ADDRSLT2.IC:=0;
613 |                       FTHADGIV.ADDRSLT2.OPR:=0
614 |               END;(*BB1 FALSE*)
615 |            END;(*AA2 FALSE*)
616 |           END(*AA1*)
617 |             ELSE
618 |           BEGIN(*AA1 FALSE*)
619 |            K:=COUNTICNUM(.J.);
620 |             IF (K<= MAXIC)THEN
621 |              BEGIN (*BB1*)
622 |               COUNTICNUM(.J.):=COUNTICNUM(.J.)+1;
623 |              IF(((CELLBLOCK(.J,K.).OPCODE)=SQURT)OR ((CELLBLOCK(.J,K.).OPCODE
624 |                 =SQAR)))THEN
625 |                  BEGIN(*BB2*)
626 |                   IF((CELLBLOCK(.J,K.).OPRAND1 <> 99))THEN
627 |                    BEGIN(*BB3*)
628 |                     IF(K<>FETCHREG(.J.))THEN
629 |                      BEGIN(*BB4*)
630 |                       FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
631 |                       FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
632 |                       FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
633 |                       FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
634 |                       FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
635 |                       FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
636 |                       FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
637 |                       FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
638 |                       FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
639 |                       FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
640 |                       FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
641 |                       FETCHREG(.J.):=K
642 |                      END(*BB4*)
643 |                     ELSE
644 |                      BEGIN(*BB4 FLASE*)
645 |                       FTHADGIV.OPCODE:=NOP;
```

```
                    FTHADGIV.OPRAND2:=0;
                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                    FTHADGIV.STAR:='A';
                    FTHADGIV.ADDRSLT1.CB:=0;
                    FTHADGIV.ADDRSLT1.IC:=0;
                    FTHADGIV.ADDRSLT1.OPR:=0;
                    FTHADGIV.ADDRSLT2.CB:=0;
                    FTHADGIV.ADDRSLT2.IC:=0;
                    FTHADGIV.ADDRSLT2.OPR:=0
                  END;(*BB4 FALSE*)
                END(*BB3*)
               ELSE
                BEGIN(*BB3 FALSE*)
                    FTHADGIV.OPCODE:=NOP;
                    FTHADGIV.OPRAND1:=0;
                    FTHADGIV.OPRAND2:=0;
                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                    FTHADGIV.STAR:='A';
                    FTHADGIV.ADDRSLT1.CB:=0;
                    FTHADGIV.ADDRSLT1.IC:=0;
                    FTHADGIV.ADDRSLT1.OPR:=0;
                    FTHADGIV.ADDRSLT2.CB:=0;
                    FTHADGIV.ADDRSLT2.IC:=0;
                    FTHADGIV.ADDRSLT2.OPR:=0
                END;(*BB3 FALSE*)
              END(*BB2*)
             ELSE(*BB2 FALSE*)
              BEGIN(*BB2F1*)
      IF((CELLBLOCK(.J,K.).OPRAND1<>99)AND(CELLBLOCK(.J,K.).OPRAND2<>99)) THEN
              BEGIN(**BB2 F2*)
               IF(K<>FETCHREG(.J.))THEN
                BEGIN(**BB2 F3*)
                    FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
                    FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
                    FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                    FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
                    FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
                    FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
                    FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
                    FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
                    FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
                    FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
                    FETCHREG(.J.):=K
                END(*BB2 F3*)
               ELSE(*BB2 F3 FLASE*)
                BEGIN(*BB2 F3 FALSE*)
                    FTHADGIV.OPCODE:=NOP;
                    FTHADGIV.OPRAND1:=0;
                    FTHADGIV.OPRAND2:=0;
                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                    FTHADGIV.STAR:='A';
                    FTHADGIV.ADDRSLT1.CB:=0;
                    FTHADGIV.ADDRSLT1.IC:=0;
                    FTHADGIV.ADDRSLT1.OPR:=0;
                    FTHADGIV.ADDRSLT2.CB:=0;
                    FTHADGIV.ADDRSLT2.IC:=0;
                    FTHADGIV.ADDRSLT2.OPR:=0
                END;(*BB2 F3 FALSE*)
              END(*BB2 F2*)
             ELSE(*BB2 F2 FALSE*)
              BEGIN(*BB2 F2 FALSE*)
                    FTHADGIV.OPCODE:=NOP;
                    FTHADGIV.OPRAND1:=0;
```

```
                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                    FTHADGIV.STAR:='A';
                    FTHADGIV.ADDRSLT1.CB:=0;
                    FTHADGIV.ADDRSLT1.IC:=0;
                    FTHADGIV.ADDRSLT1.OPR:=0;
                    FTHADGIV.ADDRSLT2.CB:=0;
                    FTHADGIV.ADDRSLT2.IC:=0;
                    FTHADGIV.ADDRSLT2.OPR:=0
          END;(*BB2 F2 FALSE*)
         END;(*BB2 F1*)
        END(*BB1*)
        ELSE(*BB1 FALSE*)
         BEGIN(*BB1 FALSE*)
                FTHADGIV.OPCODE:=NOP;
                FTHADGIV.OPRAND1:=0;
                FTHADGIV.OPRAND2:=0;
                FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                FTHADGIV.STAR:='A';
                FTHADGIV.ADDRSLT1.CB:=0;
                FTHADGIV.ADDRSLT1.IC:=0;
                FTHADGIV.ADDRSLT1.OPR:=0;
                FTHADGIV.ADDRSLT2.CB:=0;
                FTHADGIV.ADDRSLT2.IC:=0;
                FTHADGIV.ADDRSLT2.OPR:=0
          END;(*BB1 FALSE*)
       END;(*AA1 FALSE*)
           END;(*AA FALSE*)
        END;(*NOT EQ FIN*)
     END(*TRUE UPDATE YES*)
    ELSE(*IF NOT UPDATEYES TRUE*)
     BEGIN(*BB*)
      K:=COUNTICNUM(.J.);
       IF (K<= MAXIC)THEN
        BEGIN (*BB1*)
         COUNTICNUM(.J.):=COUNTICNUM(.J.)+1;
       IF(((CELLBLOCK(.J,K.).OPCODE)=SQURT)OR ((CELLBLOCK(.J,K.).OPCODE
          =SQAR)))THEN
           BEGIN(*BB2*)
            IF((CELLBLOCK(.J,K.).OPRAND1 <> 99))THEN
             BEGIN(*BB3*)
              IF(K<>FETCHREG(.J.))THEN
               BEGIN(*BB4*)
                FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
                FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
                FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
                FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
                FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
                FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
                FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
                FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
                FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
                FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
                FETCHREG(.J.):=K
               END(*BB4*)
              ELSE
               BEGIN(*BB4 FLASE*)
                FTHADGIV.OPCODE:=NOP;
                FTHADGIV.OPRAND1:=0;
                FTHADGIV.OPRAND2:=0;
                FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
                FTHADGIV.STAR:='A';
                FTHADGIV.ADDRSLT1.CB:=0;
                FTHADGIV.ADDRSLT1.IC:=0;
```

```
777 |                    FTHADGIV.ADDRSLT2.CB:=0;
778 |                    FTHADGIV.ADDRSLT2.IC:=0;
779 |                    FTHADGIV.ADDRSLT2.OPR:=0
780 |                  END;(*BB4 FALSE*)
781 |                END(*BB3*)
782 |               ELSE
783 |                BEGIN(*BB3 FALSE*)
784 |                    FTHADGIV.OPCODE:=NOP;
785 |                    FTHADGIV.OPRAND1:=0;
786 |                    FTHADGIV.OPRAND2:=0;
787 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
788 |                    FTHADGIV.STAR:='A';
789 |                    FTHADGIV.ADDRSLT1.CB:=0;
790 |                    FTHADGIV.ADDRSLT1.IC:=0;
791 |                    FTHADGIV.ADDRSLT1.OPR:=0;
792 |                    FTHADGIV.ADDRSLT2.CB:=0;
793 |                    FTHADGIV.ADDRSLT2.IC:=0;
794 |                    FTHADGIV.ADDRSLT2.OPR:=0
795 |                END;(*BB3 FALSE*)
796 |              END(*BB2*)
797 |             ELSE(*BB2 FALSE*)
798 |              BEGIN(*BB2F1*)
799 |           IF((CELLBLOCK(.J,K.).OPRAND1<>99)AND(CELLBLOCK(.J,K.).OPRAND2<>99)) THEN
800 |              BEGIN(**BB2 F2*)
801 |               IF(K<>FETCHREG(.J.))THEN
802 |                BEGIN(**BB2 F3*)
803 |                    FTHADGIV.OPCODE:=CELLBLOCK(.J,K.).OPCODE;
804 |                    FTHADGIV.OPRAND1:=CELLBLOCK(.J,K.).OPRAND1;
805 |                    FTHADGIV.OPRAND2:=CELLBLOCK(.J,K.).OPRAND2;
806 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
807 |                    FTHADGIV.STAR:=CELLBLOCK(.J,K.).STAR;
808 |                    FTHADGIV.ADDRSLT1.CB:=CELLBLOCK(.J,K.).ADDRSLT1.CB;
809 |                    FTHADGIV.ADDRSLT1.IC:=CELLBLOCK(.J,K.).ADDRSLT1.IC;
810 |                    FTHADGIV.ADDRSLT1.OPR:=CELLBLOCK(.J,K.).ADDRSLT1.OPR;
811 |                    FTHADGIV.ADDRSLT2.CB:=CELLBLOCK(.J,K.).ADDRSLT2.CB;
812 |                    FTHADGIV.ADDRSLT2.IC:=CELLBLOCK(.J,K.).ADDRSLT2.IC;
813 |                    FTHADGIV.ADDRSLT2.OPR:=CELLBLOCK(.J,K.).ADDRSLT2.OPR;
814 |                    FETCHREG(.J.):=K
815 |                END(*BB2 F3*)
816 |               ELSE(*BB2 F3 FLASE*)
817 |                BEGIN(*BB2 F3 FALSE*)
818 |                    FTHADGIV.OPCODE:=NOP;
819 |                    FTHADGIV.OPRAND1:=0;
820 |                    FTHADGIV.OPRAND2:=0;
821 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
822 |                    FTHADGIV.STAR:='A';
823 |                    FTHADGIV.ADDRSLT1.CB:=0;
824 |                    FTHADGIV.ADDRSLT1.IC:=0;
825 |                    FTHADGIV.ADDRSLT1.OPR:=0;
826 |                    FTHADGIV.ADDRSLT2.CB:=0;
827 |                    FTHADGIV.ADDRSLT2.IC:=0;
828 |                    FTHADGIV.ADDRSLT2.OPR:=0
829 |                END;(*BB2 F3 FALSE*)
830 |             END(*BB2 F2*)
831 |            ELSE(*BB2 F2 FALSE*)
832 |             BEGIN(*BB2 F2 FALSE*)
833 |                    FTHADGIV.OPCODE:=NOP;
834 |                    FTHADGIV.OPRAND1:=0;
835 |                    FTHADGIV.OPRAND2:=0;
836 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
837 |                    FTHADGIV.STAR:='A';
838 |                    FTHADGIV.ADDRSLT1.CB:=0;
839 |                    FTHADGIV.ADDRSLT1.IC:=0;
840 |                    FTHADGIV.ADDRSLT1.OPR:=0;
```

```
842 |                    FTHADGIV.ADDRSLT2.IC:=0;
843 |                    FTHADGIV.ADDRSLT2.OPR:=0
844 |              END;(*BB2 F2 FALSE*)
845 |             END;(*BB2 F1*)
846 |            END(*BB1*)
847 |            ELSE(*BB1 FALSE*)
848 |             BEGIN(*BB1 FALSE*)
849 |                    FTHADGIV.OPCODE:=NOP;
850 |                    FTHADGIV.OPRAND1:=0;
851 |                    FTHADGIV.OPRAND2:=0;
852 |                    FTHADGIV.RESULT:=99;(*LEAVING IT BLNK TO BE FILLED BY PE*)
853 |                    FTHADGIV.STAR:='A';
854 |                    FTHADGIV.ADDRSLT1.CB:=0;
855 |                    FTHADGIV.ADDRSLT1.IC:=0;
856 |                    FTHADGIV.ADDRSLT1.OPR:=0;
857 |                    FTHADGIV.ADDRSLT2.CB:=0;
858 |                    FTHADGIV.ADDRSLT2.IC:=0;
859 |                    FTHADGIV.ADDRSLT2.OPR:=0
860 |             END;(*BB1 FALSE*)
861 |            END;(*BB*)
862 |           END;(*PROCEDURE FETCHUNIT*)
863 |   END;(*111*)
864 |   (**********************PROCEDURE FOR PROCESSING ELEMENT***********)
865 |   (***************ALL VARIABLES ARE GLOBAL*******************)
866 |   (***************ISSUE:CODING OF BR:-SINCE BR IS INSIDE THE INTEGER*)
867 |   (*REPRESENTATION 0 0 0 0 0 BR THEREFORE BR=0 IS REPRESENTED AS    *)
868 |   (**********BR=0=>999, BR=1=>9999************************************)
869 |   PROCEDURE PROCELEM;
870 |   (*LOCAL VARIABLES,STORE,STOR,VARX=OPRAND1,VARY=OPRAND2,VARZ=RESULT*)
871 |   CONST
872 |    CON1=999;
873 |    CON2=9999;
874 |   VAR
875 |    VOPCOD:CODE;
876 |    STORE:STARVAL;
877 |    VARX,VARY,VARZ,PIPINDX,STOR:INTEGER;
878 |   BEGIN
879 |      VARX:=FTHADGIV.OPRAND1;
880 |      VARY:=FTHADGIV.OPRAND2;
881 |      VOPCOD:=FTHADGIV.OPCODE;
882 |        CASE VOPCOD OF
883 |          NOP:
884 |              VARZ:=0;
885 |          ADD:
886 |              VARZ:=VARX+VARY;
887 |          SUB:
888 |              VARZ:=VARX-VARY;
889 |              (***NOTE OPRAND1-OPRAND2**)
890 |          MUL:
891 |              VARZ:=VARX*VARY;
892 |          DIVI:
893 |              VARZ:=VARX DIV VARY;
894 |   (******NOTE:SINGLE VALUE OPERATIONS WILL TAKE OPRAND1********)
895 |          SQURT:
896 |              VARZ:=ROUND(SQRT(VARX));
897 |          SQAR:
898 |              VARZ:=SQR(VARX);
899 |          COMP:
900 |              IF (VARX=VARY)THEN
901 |                  VARZ:=CON1 (*I.E OPR1=FINVAL,BR=0*)
902 |               ELSE
903 |                  VARZ:=CON2;(*IE OPR1 NOT EQUAL TO FINVAL,BR=1*)
904 |          LOOPI:
905 |                BEGIN(*JUST FOLLOWING SIMU. BREAKDOWN*)
```

```
907 |  (*RESULT PACKET BEING SENT TO THE DISTRIBUTION NETWORK*)
908 |                  VARZ:=VARY+0;
909 |                  IF (VARX=999)THEN FTHADGIV.STAR:='C';
910 |                  IF (VARX=9999)THEN FTHADGIV.STAR:='D'
911 |                END;
912 |            FIN:
913 |                 VARZ:=0
914 |       END;(*CASE*)
915 |  (********PUTTING THE RESULT IN THE ARRAY AND FORMING THE ***)
916 |  (*RESULT PACKET***************)
917 |  (***PIPELINE SIMULATION*OF RESULT*******************)
918 |        FOR PIPINDX:=1 TO PIPELENGTH DO
919 |          BEGIN
920 |           STOR:=PROCCALC(.J,PIPINDX.).RESULT;
921 |           PROCCALC(.J,PIPINDX.).RESULT:=VARZ;
922 |           VARZ:=STOR
923 |          END;
924 |  (*REMEMBER WHY USING J:BECAUSE J IS COMING FROM THE FETCH UNIT*)
925 |  (* BECAUSE RIGHT AWAY AFTER FETCH UNIT THE PR.ELEMTNET IS CALLED*)
926 |  (*REMEMBER THE 1TO 1 CORR TO CB AND PE**)
927 |  (* NOTICE THAT THE J REPRESENTS THE CELLBLOCK****)
928 |  (*************************************************************)
929 |  (*********NOW SIMULATING THE ADDRESS PIPELING**************)
930 |  (************WARNING WATCH OUT FOR THE J************)
931 |  (*********B E W A R E*********************************)
932 |        FOR PIPINDX:=1 TO PIPELENGTH DO
933 |          BEGIN
934 |           STOR:=PROCADDRESS(.J,PIPINDX.).ADDRSLT1.CB;
935 |           PROCADDRESS(.J,PIPINDX.).ADDRSLT1.CB:=FTHADGIV.ADDRSLT1.CB;
936 |           FTHADGIV.ADDRSLT1.CB:=STOR
937 |          END;
938 |  (******************NEXT ADDRESS FIELD PIPELINE*****************)
939 |        FOR PIPINDX:=1 TO PIPELENGTH DO
940 |          BEGIN
941 |           STOR:=PROCADDRESS(.J,PIPINDX.).ADDRSLT1.IC;
942 |           PROCADDRESS(.J,PIPINDX.).ADDRSLT1.IC:=FTHADGIV.ADDRSLT1.IC;
943 |           FTHADGIV.ADDRSLT1.IC:=STOR
944 |          END;
945 |  (*************************************************************)
946 |        FOR PIPINDX:=1 TO PIPELENGTH DO
947 |          BEGIN
948 |           STOR:=PROCADDRESS(.J,PIPINDX.).ADDRSLT1.OPR;
949 |           PROCADDRESS(.J,PIPINDX.).ADDRSLT1.OPR:=FTHADGIV.ADDRSLT1.OPR;
950 |           FTHADGIV.ADDRSLT1.OPR:=STOR
951 |          END;
952 |  (*************************************************************)
953 |        FOR PIPINDX:=1 TO PIPELENGTH DO
954 |          BEGIN
955 |           STOR:=PROCADDRESS(.J,PIPINDX.).ADDRSLT2.CB;
956 |           PROCADDRESS(.J,PIPINDX.).ADDRSLT2.CB:=FTHADGIV.ADDRSLT2.CB;
957 |           FTHADGIV.ADDRSLT2.CB:=STOR
958 |          END;
959 |  (******************NEXT ADDRESS FIELD PIPELINE*****************)
960 |        FOR PIPINDX:=1 TO PIPELENGTH DO
961 |          BEGIN
962 |           STOR:=PROCADDRESS(.J,PIPINDX.).ADDRSLT2.IC;
963 |           PROCADDRESS(.J,PIPINDX.).ADDRSLT2.IC:=FTHADGIV.ADDRSLT2.IC;
964 |           FTHADGIV.ADDRSLT2.IC:=STOR
965 |          END;
966 |  (*************************************************************)
967 |        FOR PIPINDX:=1 TO PIPELENGTH DO
968 |          BEGIN
969 |           STOR:=PROCADDRESS(.J,PIPINDX.).ADDRSLT2.OPR;
970 |           PROCADDRESS(.J,PIPINDX.).ADDRSLT2.OPR:=FTHADGIV.ADDRSLT2.OPR;
```

```
            END;
  (**********************NOW THE STAR FIELD PIPLINE*********************)
        FOR PIPINDX:=1 TO PIPELENGTH DO
          BEGIN
           STORE:=PROCADDRESS(.J,PIPINDX.).STAR;
           PROCADDRESS(.J,PIPINDX.).STAR:=FTHADGIV.STAR;
           FTHADGIV.STAR:=STORE
          END;
  (*************************************************************************)
  (*********************FORMING THE RESULT PACKET************************)
        RESLTPAC.RESULT:=PROCCALC(.J,PIPELENGTH.).RESULT;
        RESLTPAC.STAR:=PROCADDRESS(.J,PIPELENGTH.).STAR;
        RESLTPAC.ADDRSLT1.CB:=PROCADDRESS(.J,PIPELENGTH.).ADDRSLT1.CB;
        RESLTPAC.ADDRSLT1.IC:=PROCADDRESS(.J,PIPELENGTH.).ADDRSLT1.IC;
        RESLTPAC.ADDRSLT1.OPR:=PROCADDRESS(.J,PIPELENGTH.).ADDRSLT1.OPR;
        RESLTPAC.ADDRSLT2.CB:=PROCADDRESS(.J,PIPELENGTH.).ADDRSLT2.CB;
        RESLTPAC.ADDRSLT2.IC:=PROCADDRESS(.J,PIPELENGTH.).ADDRSLT2.IC;
        RESLTPAC.ADDRSLT2.OPR:=PROCADDRESS(.J,PIPELENGTH.).ADDRSLT2.OPR
  END;(*PROCESSING ELEMENT*)
  (*************************************************************************)
  (*************************************************************************)
  (*****************************PROCEDURE DISTRIBUTION NETWORK******)
  PROCEDURE DISTNET;
  (************************LOCAL VARIABLE DEFINED****************)
  VAR
    LSTAR:STARVAL;
  BEGIN
  (****STAR NEEDS TO BE TRANSMITTED AS IT IS TO THE UPDATE UNIT SO***)
     DISADDGIV.STAR:=RESLTPAC.STAR;
  (**********************SAME WITH RESULT*********************)
      DISADDGIV.RESULT:=RESLTPAC.RESULT;
      LSTAR:=RESLTPAC.STAR;
        CASE LSTAR OF
         'A':(* *=00,EXAMINE ADDRSLT1 TO SEND TO CORR.CB*)
             BEGIN
               DISADDGIV.ADDRSLT1.CB:=RESLTPAC.ADDRSLT1.CB;
               DISADDGIV.ADDRSLT1.IC:=RESLTPAC.ADDRSLT1.IC;
               DISADDGIV.ADDRSLT1.OPR:=RESLTPAC.ADDRSLT1.OPR;
  (***************PUTTING THE ADDRSLT2 TO BE 0*****************)
               DISADDGIV.ADDRSLT2.CB:=0;
               DISADDGIV.ADDRSLT2.IC:=0;
               DISADDGIV.ADDRSLT2.OPR:=0
             END;
         'B':(* *=01,BOTH ADDRSLT1 AND ADDRSLT2 ARE VALID*)
             (**BOTH NEED TO BE SEND TO CORROSPONDING CB AND THEN*)
             (*TO THE UPDATE UNIT. UPDATE UNIT WILL POINT THE CONFLICT*)
             (* IF THE UPDATING IS DONE WITHIN THE SAME CELL BLOCK**)
             BEGIN
               DISADDGIV.ADDRSLT1.CB:=RESLTPAC.ADDRSLT1.CB;
               DISADDGIV.ADDRSLT1.IC:=RESLTPAC.ADDRSLT1.IC;
               DISADDGIV.ADDRSLT1.OPR:=RESLTPAC.ADDRSLT1.OPR;
               DISADDGIV.ADDRSLT2.CB:=RESLTPAC.ADDRSLT2.CB;
               DISADDGIV.ADDRSLT2.IC:=RESLTPAC.ADDRSLT2.IC;
               DISADDGIV.ADDRSLT2.OPR:=RESLTPAC.ADDRSLT2.OPR
             END;
         'C':(* *=10 I.E TAKE ADDRESS RESLULT 1 TO CORR. CB*)
             (****I.E  SAME AS 'A'**,THIS WILL BE USED FOR LOOP USING BR*)
             BEGIN
               DISADDGIV.ADDRSLT1.CB:=RESLTPAC.ADDRSLT1.CB;
               DISADDGIV.ADDRSLT1.IC:=RESLTPAC.ADDRSLT1.IC;
               DISADDGIV.ADDRSLT1.OPR:=RESLTPAC.ADDRSLT1.OPR;
  (***************PUTTING THE ADDRSLT2 TO BE 0*****************)
               DISADDGIV.ADDRSLT2.CB:=0;
               DISADDGIV.ADDRSLT2.IC:=0;
```

```
1037 |                 END;
1038 |          'D':(** *=11,THIS USED FOR LOOPING USING BR*)
1039 |              BEGIN
1040 |                DISADDGIV.ADDRSLT2.CB:=RESLTPAC.ADDRSLT2.CB;
1041 |                DISADDGIV.ADDRSLT2.IC:=RESLTPAC.ADDRSLT2.IC;
1042 |                DISADDGIV.ADDRSLT2.OPR:=RESLTPAC.ADDRSLT2.OPR;
1043 |  (***************PUTTING THE ADDRSLT2 TO BE 0*****************)
1044 |                DISADDGIV.ADDRSLT1.CB:=0;
1045 |                DISADDGIV.ADDRSLT1.IC:=0;
1046 |                DISADDGIV.ADDRSLT1.OPR:=0
1047 |              END
1048 |            END(*CASE*)
1049 |  END;(* END PROCEDURE DISTRIBUTION NETWORK*)
1050 |  (*************************************************************)
1051 |  (**THE ONLY UNIT WHICH CAN WRITE IN THE MEMORY, THE ADDRESS IS *)
1052 |  (*GIVEN ONLY OF THE IC DEPENDING ON CB ALREADY FOUND BY THE DIS.NETW*)
1053 |  PROCEDURE UPDATE;
1054 |  VAR
1055 |    LLSTAR:STARVAL;
1056 |    VK,VI,VL,STOR,VARJ,VARK,VOPR,VARX,VGG:INTEGER;
1057 |    VB,VSTOR:BOOLEAN;
1058 |  BEGIN(*NOTICE YOU CAN REPLACE LOTS OF CODE BY SINGLE PROC. BUT IT*)
1059 |       (*IS AVOIDED BECAUSE TO REPRESENT THE FUNCTIONALITY OF UNITS*)
1060 |    LLSTAR:=DISADDGIV.STAR;
1061 |      CASE LLSTAR OF
1062 |       'A':
1063 |          BEGIN
1064 |           VK:=DISADDGIV.ADDRSLT1.CB;
1065 |           IF
1066 |            (((DISADDGIV.ADDRSLT1.CB)=0)AND((DISADDGIV.ADDRSLT1.IC)=0)AND
1067 |            ((DISADDGIV.ADDRSLT1.OPR)=0))THEN
1068 |  (*************************************************************)
1069 |             BEGIN(*FALSE*)
1070 |               IF SIMSAVB(.CBLOOPVAL.) THEN
1071 |                BEGIN(*2*)
1072 |                 VGG:=SIMSAVE(.CBLOOPVAL.);
1073 |                 FOR VI:=1 TO MAXCELBLK DO
1074 |                  BEGIN
1075 |                   VSTOR:=UPDATYES(.VGG,VI.);
1076 |                   UPDATYES(.VGG,VI.):=SIMSAVB(.CBLOOPVAL.);
1077 |                   SIMSAVB(.CBLOOPVAL.):=VSTOR
1078 |                  END;
1079 |                FOR VI:=1 TO MAXCELBLK DO
1080 |                 BEGIN
1081 |                  STOR:=UPADDGIV(.VGG,VI.).IC;
1082 |                  UPADDGIV(.VGG,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1083 |                  SIMSAVE(.CBLOOPVAL.):=STOR
1084 |                 END;
1085 |                 SIMSAVE(.CBLOOPVAL.):=0;
1086 |                 SIMSAVB(.CBLOOPVAL.):=FALSE
1087 |               END;(*2*)
1088 |             VB:=FALSE;
1089 |             FOR VI:= 1 TO MAXCELBLK DO
1090 |              BEGIN(*1*)
1091 |               VSTOR:=UPDATYES(.CBLOOPVAL,VI.);
1092 |               UPDATYES(.CBLOOPVAL,VI.):=VB;
1093 |               VB:=VSTOR
1094 |              END;(*1*)
1095 |               VARX:=0;
1096 |               FOR VI:=1 TO MAXCELBLK DO
1097 |                BEGIN
1098 |                 STOR:=UPADDGIV(.CBLOOPVAL,VI.).IC;
1099 |                 UPADDGIV(.CBLOOPVAL,VI.).IC:=VARX;
1100 |                 VARX:=STOR
```

```
1102 |              END(*FALSE*)
1103 |            ELSE
1104 |             BEGIN(*1A*)
1105 |             (*GIVING THE ADDRESS VALUE TO THE FETHC UNIT,FOR*)
1106 |             (*THE INSTRUCTION JUST UPDATED*)
1107 |              IF (VK > CBLOOPVAL) THEN(*BECAUSE FIRST TIME AROUND IT WOULD*)
1108 |                                       (*BE FALSE**************************)
1109 |               BEGIN(*1*)
1110 |                IF SIMSAVB(.CBLOOPVAL.) THEN
1111 |                 BEGIN(*2*)
1112 |                  FOR VI:=1 TO MAXCELBLK DO
1113 |                   BEGIN
1114 |                    VSTOR:=UPDATYES(.VK,VI.);
1115 |                    UPDATYES(.VK,VI.):=SIMSAVB(.CBLOOPVAL.);
1116 |                    SIMSAVB(.CBLOOPVAL.):=VSTOR
1117 |                   END;
1118 |                  FOR VI:=1 TO MAXCELBLK DO
1119 |                   BEGIN
1120 |                    STOR:=UPADDGIV(.VK,VI.).IC;
1121 |                    UPADDGIV(.VK,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1122 |                    SIMSAVE(.CBLOOPVAL.):=STOR
1123 |                   END;
1124 |                  SIMSAVE(.CBLOOPVAL.):=VK;(*STORING FOR NEXT USE*)
1125 |                  SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1126 |                 END(*2*)
1127 |               ELSE(*IF SIMSAVB(.CBLOOPVAL.) IS FALSE*)
1128 |                (*JUST STORE FOR THE NEXT VALUE*)
1129 |                   SIMSAVE(.CBLOOPVAL.):=VK;(*STORING FOR NEXT USE*)
1130 |                   SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1131 |            END(*1*)
1132 |          ELSE(*IF VK<= TO CBLOOP VAL*)
1133 |           BEGIN(*3*)
1134 |                 IF SIMSAVB(.CBLOOPVAL.) THEN
1135 |                  BEGIN(*2*)
1136 |                   VGG:=SIMSAVE(.CBLOOPVAL.);
1137 |                   FOR VI:=1 TO MAXCELBLK DO
1138 |                    BEGIN
1139 |                     VSTOR:=UPDATYES(.VGG,VI.);
1140 |                     UPDATYES(.VGG,VI.):=SIMSAVB(.CBLOOPVAL.);
1141 |                     SIMSAVB(.CBLOOPVAL.):=VSTOR
1142 |                    END;
1143 |                   FOR VI:=1 TO MAXCELBLK DO
1144 |                    BEGIN
1145 |                     STOR:=UPADDGIV(.VGG,VI.).IC;
1146 |                     UPADDGIV(.VGG,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1147 |                     SIMSAVE(.CBLOOPVAL.):=STOR
1148 |                    END;
1149 |                   SIMSAVE(.CBLOOPVAL.):=0;
1150 |                   SIMSAVB(.CBLOOPVAL.):=FALSE
1151 |                  END;(*2*)
1152 |                VB:=TRUE;
1153 |                FOR VI:=1 TO MAXCELBLK DO
1154 |                 BEGIN
1155 |                  VSTOR:=UPDATYES(.VK,VI.);
1156 |                  UPDATYES(.VK,VI.):=VB;
1157 |                  VB:=VSTOR
1158 |                 END;
1159 |                VARX:=DISADDGIV.ADDRSLT1.IC;
1160 |                 FOR VI:=1 TO MAXCELBLK DO
1161 |                   BEGIN(*SAVING THE ADDRESS UPDATED BY UDU FOR F.U*)
1162 |                    STOR:=UPADDGIV(.VK,VI.).IC;
1163 |                    UPADDGIV(.VK,VI.).IC:=VARX;
1164 |                    VARX:=STOR
1165 |                   END;
```

```
1167 |  (****************NEXT UPDATING THE CELL BLOCK*****************)
1168 |  (**************ACTUALLY WRITTING IN THE ARRY*WITH RESULT******)
1169 |              VOPR:=DISADDGIV.ADDRSLT1.OPR;
1170 |              VARJ:=DISADDGIV.ADDRSLT1.CB;
1171 |              VARK:=DISADDGIV.ADDRSLT1.IC;
1172 |               IF (VOPR=1)THEN
1173 |                CELLBLOCK(.VARJ,VARK.).OPRAND1:=DISADDGIV.RESULT;
1174 |               IF (VOPR=2)THEN
1175 |                CELLBLOCK(.VARJ,VARK.).OPRAND2:=DISADDGIV.RESULT
1176 |          END(*1A*)
1177 |         END;(*END BEGIN*)
1178 |         'B':
1179 |
1180 |           BEGIN
1181 |            IF
1182 |            ((((DISADDGIV.ADDRSLT1.CB)=0)AND((DISADDGIV.ADDRSLT1.IC)=0)AND
1183 |             ((DISADDGIV.ADDRSLT1.OPR)=0))AND
1184 |             (((DISADDGIV.ADDRSLT2.CB)=0)AND((DISADDGIV.ADDRSLT2.IC)=0)AND
1185 |             ((DISADDGIV.ADDRSLT2.OPR)=0)))THEN
1186 |  (*************************************************************)
1187 |              BEGIN(*FALSE *)
1188 |                IF SIMSAVB(.CBLOOPVAL.) THEN
1189 |                 BEGIN(*2*)
1190 |                ' VGG:=SIMSAVE(.CBLOOPVAL.);
1191 |                  FOR VI:=1 TO MAXCELBLK DO
1192 |                   BEGIN
1193 |                    VSTOR:=UPDATYES(.VGG,VI.);
1194 |                    UPDATYES(.VGG,VI.):=SIMSAVB(.CBLOOPVAL.);
1195 |                    SIMSAVB(.CBLOOPVAL.):=VSTOR
1196 |                   END;
1197 |                  FOR VI:=1 TO MAXCELBLK DO
1198 |                   BEGIN
1199 |                    STOR:=UPADDGIV(.VGG,VI.).IC;
1200 |                    UPADDGIV(.VGG,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1201 |                    SIMSAVE(.CBLOOPVAL.):=STOR
1202 |                   END;
1203 |                   SIMSAVE(.CBLOOPVAL.):=0;
1204 |                   SIMSAVB(.CBLOOPVAL.):=FALSE
1205 |                 END;(*2*)
1206 |               VB:=FALSE;
1207 |               FOR VI:= 1 TO MAXCELBLK DO
1208 |                BEGIN(*1*)
1209 |                 VSTOR:=UPDATYES(.CBLOOPVAL,VI.);
1210 |                 UPDATYES(.CBLOOPVAL,VI.):=VB;
1211 |                 VB:=VSTOR
1212 |                END;(*1*)
1213 |                 VARX:=0;
1214 |                 FOR VI:=1 TO MAXCELBLK DO
1215 |                  BEGIN
1216 |                   STOR:=UPADDGIV(.CBLOOPVAL,VI.).IC;
1217 |                   UPADDGIV(.CBLOOPVAL,VI.).IC:=VARX;
1218 |                   VARX:=STOR
1219 |                  END;
1220 |             END(*FALSE*)
1221 |           ELSE
1222 |         BEGIN(*1A*)
1223 |            (*GIVING THE ADDRESS VALUE TO THE FETHC UNIT,FOR*)
1224 |                  (*THE INSTRUCTION JUST UPDATED*)
1225 |  (*******ALTHOUGH BOTH THE INSTRUCTIONS WILL BE UPDATED I.E WRITTEN*)
1226 |  (**ON BUT TO THE FETCH UNIT ONLY ONE ADDRESS WILL BE SENT**I.E ADDRESLT1*)
1227 |              VK:=DISADDGIV.ADDRSLT1.CB;
1228 |              IF (VK > CBLOOPVAL) THEN(*BECAUSE FIRST TIME AROUND IT WOULD*)
1229 |                                      (*BE FALSE**************************)
1230 |               BEGIN(*1*)
```

```
1232 |                  BEGIN(*2*)
1233 |                   FOR VI:=1 TO MAXCELBLK DO
1234 |                    BEGIN
1235 |                     VSTOR:=UPDATYES(.VK,VI.);
1236 |                     UPDATYES(.VK,VI.):=SIMSAVB(.CBLOOPVAL.);
1237 |                     SIMSAVB(.CBLOOPVAL.):=VSTOR
1238 |                    END;
1239 |                   FOR VI:=1 TO MAXCELBLK DO
1240 |                    BEGIN
1241 |                     STOR:=UPADDGIV(.VK,VI.).IC;
1242 |                     UPADDGIV(.VK,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1243 |                     SIMSAVE(.CBLOOPVAL.):=STOR
1244 |                    END;
1245 |                  SIMSAVE(.CBLOOPVAL.):=VK;(*STORING FOR NEXT USE*)
1246 |                  SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1247 |                END(*2*)
1248 |               ELSE(*IF SIMSAVB(.CBLOOPVAL.) IS FALSE*)
1249 |                (*JUST STORE FOR THE NEXT VALUE*)
1250 |                  SIMSAVE(.CBLOOPVAL.):=VK;(*STORING FOR NEXT USE*)
1251 |                  SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1252 |            END(*1*)
1253 |          ELSE(*IF VK<= TO CBLOOP VAL*)
1254 |           BEGIN(*3*)
1255 |                 IF SIMSAVB(.CBLOOPVAL.) THEN
1256 |                  BEGIN(*2*)
1257 |                   VGG:=SIMSAVE(.CBLOOPVAL.);
1258 |                   FOR VI:=1 TO MAXCELBLK DO
1259 |                    BEGIN
1260 |                     VSTOR:=UPDATYES(.VGG,VI.);
1261 |                     UPDATYES(.VGG,VI.):=SIMSAVB(.CBLOOPVAL.);
1262 |                     SIMSAVB(.CBLOOPVAL.):=VSTOR
1263 |                    END;
1264 |                   FOR VI:=1 TO MAXCELBLK DO
1265 |                    BEGIN
1266 |                     STOR:=UPADDGIV(.VGG,VI.).IC;
1267 |                     UPADDGIV(.VGG,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1268 |                     SIMSAVE(.CBLOOPVAL.):=STOR
1269 |                    END;
1270 |                   SIMSAVE(.CBLOOPVAL.):=0;
1271 |                   SIMSAVB(.CBLOOPVAL.):=FALSE
1272 |                 END;(*2*)
1273 |              VB:=TRUE;
1274 |              FOR VI:=1 TO MAXCELBLK DO
1275 |               BEGIN
1276 |                VSTOR:=UPDATYES(.VK,VI.);
1277 |                UPDATYES(.VK,VI.):=VB;
1278 |                VB:=VSTOR
1279 |               END;
1280 |              VARX:=DISADDGIV.ADDRSLT1.IC;
1281 |               FOR VI:=1 TO MAXCELBLK DO
1282 |                BEGIN
1283 |                 STOR:=UPADDGIV(.VK,VI.).IC;
1284 |                 UPADDGIV(.VK,VI.).IC:=VARX;
1285 |                 VARX:=STOR
1286 |                END;
1287 |           END;(*3*)
1288 |             (**ADDRSLT2*)
1289 |              VL:=DISADDGIV.ADDRSLT2.CB;
1290 |             (*GIVING THE ADDRESS VALUE TO THE FETHC UNIT,FOR*)
1291 |             (*THE INSTRUCTION JUST UPDATED*)
1292 |       IF (VL=VK)THEN
1293 |        BEGIN(*VL=VK*)
1294 |              VB:=TRUE;
1295 |              FOR VI:=1 TO MAXCELBLK DO
```

```
1297 |                VSTOR:=UPDATYES(.VL,VI.);
1298 |                UPDATYES(.VL,VI.):=VB;
1299 |                VB:=VSTOR
1300 |               END;
1301 |              VARX:=DISADDGIV.ADDRSLT2.IC;
1302 |               FOR VI:=1 TO MAXCELBLK DO
1303 |                BEGIN
1304 |                 STOR:=UPADDGIV(.VL,VI.).IC;
1305 |                 UPADDGIV(.VL,VI.).IC:=VARX;
1306 |                 VARX:=STOR
1307 |                END;
1308 |        END(*VL=VK*)
1309 |       ELSE
1310 |        BEGIN(*VL NOT EQUAL VK*)
1311 |              IF (VL > CBLOOPVAL) THEN(*BECAUSE FIRST TIME AROUND IT WOULD*)
1312 |                                       (*BE FALSE**************************)
1313 |               BEGIN(*1*)
1314 |                IF SIMSAVB(.CBLOOPVAL.) THEN
1315 |                 BEGIN(*2*)
1316 |                  FOR VI:=1 TO MAXCELBLK DO
1317 |                   BEGIN
1318 |                    VSTOR:=UPDATYES(.VL,VI.);
1319 |                    UPDATYES(.VL,VI.):=SIMSAVB(.CBLOOPVAL.);
1320 |                    SIMSAVB(.CBLOOPVAL.):=VSTOR
1321 |                   END;
1322 |                  FOR VI:=1 TO MAXCELBLK DO
1323 |                   BEGIN
1324 |                    STOR:=UPADDGIV(.VL,VI.).IC;
1325 |                    UPADDGIV(.VL,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1326 |                    SIMSAVE(.CBLOOPVAL.):=STOR
1327 |                   END;
1328 |                  SIMSAVE(.CBLOOPVAL.):=VL;(*STORING FOR NEXT USE*)
1329 |                  SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1330 |                 END(*2*)
1331 |                ELSE(*IF SIMSAVB(.CBLOOPVAL.) IS FALSE*)
1332 |                 (*JUST STORE FOR THE NEXT VALUE*)
1333 |                    SIMSAVE(.CBLOOPVAL.):=VL;(*STORING FOR NEXT USE*)
1334 |                    SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1335 |           END(*1*)
1336 |         ELSE(*IF VL<= TO CBLOOP VAL*)
1337 |          BEGIN(*3*)
1338 |                IF SIMSAVB(.CBLOOPVAL.) THEN
1339 |                 BEGIN(*2*)
1340 |                  VGG:=SIMSAVE(.CBLOOPVAL.);
1341 |                  FOR VI:=1 TO MAXCELBLK DO
1342 |                   BEGIN
1343 |                    VSTOR:=UPDATYES(.VGG,VI.);
1344 |                    UPDATYES(.VGG,VI.):=SIMSAVB(.CBLOOPVAL.);
1345 |                    SIMSAVB(.CBLOOPVAL.):=VSTOR
1346 |                   END;
1347 |                  FOR VI:=1 TO MAXCELBLK DO
1348 |                   BEGIN
1349 |                    STOR:=UPADDGIV(.VGG,VI.).IC;
1350 |                    UPADDGIV(.VGG,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1351 |                    SIMSAVE(.CBLOOPVAL.):=STOR
1352 |                   END;
1353 |                   SIMSAVE(.CBLOOPVAL.):=0;
1354 |                   SIMSAVB(.CBLOOPVAL.):=FALSE
1355 |                 END;(*2*)
1356 |               VB:=TRUE;
1357 |               FOR VI:=1 TO MAXCELBLK DO
1358 |                BEGIN
1359 |                 VSTOR:=UPDATYES(.VL,VI.);
1360 |                 UPDATYES(.VL,VI.):=VB;
```

```
1362                END;
1363               VARX:=DISADDGIV.ADDRSLT2.IC;
1364                FOR VI:=1 TO MAXCELBLK DO
1365                 BEGIN
1366                  STOR:=UPADDGIV(.VL,VI.).IC;
1367                  UPADDGIV(.VL,VI.).IC:=VARX;
1368                  VARX:=STOR
1369                 END;
1370             END;(*3*)
1371           END;(*VL NOT EQUAL VK*)
1372     (*****************NEXT UPDATING THE CELL BLOCK*****************)
1373     (**************ACTUALLY WRITTING IN THE ARRY*WITH RESULT******)
1374               VOPR:=DISADDGIV.ADDRSLT1.OPR;
1375               VARJ:=DISADDGIV.ADDRSLT1.CB;
1376               VARK:=DISADDGIV.ADDRSLT1.IC;
1377                IF (VOPR=1)THEN
1378                 CELLBLOCK(.VARJ,VARK.).OPRAND1:=DISADDGIV.RESULT;
1379                IF (VOPR=2)THEN
1380                 CELLBLOCK(.VARJ,VARK.).OPRAND2:=DISADDGIV.RESULT;
1381     (****GOING FOR WRITEUP OF SECOND ADDRESS*************)
1382     (****SAME RESULT WILL BE WRITTEN IN TWO PLACES********)
1383               VOPR:=DISADDGIV.ADDRSLT2.OPR;
1384               VARJ:=DISADDGIV.ADDRSLT2.CB;
1385               VARK:=DISADDGIV.ADDRSLT2.IC;
1386                IF (VOPR=1)THEN
1387                 CELLBLOCK(.VARJ,VARK.).OPRAND1:=DISADDGIV.RESULT;
1388                IF (VOPR=2)THEN
1389                 CELLBLOCK(.VARJ,VARK.).OPRAND2:=DISADDGIV.RESULT;
1390             END(*1A*)
1391           END;(*END BEGIN*)
1392         'C':(*SAME OPERATION AS 'A'*)
1393           BEGIN
1394            VK:=DISADDGIV.ADDRSLT1.CB;
1395            IF
1396             (((DISADDGIV.ADDRSLT1.CB)=0)AND((DISADDGIV.ADDRSLT1.IC)=0)AND
1397             ((DISADDGIV.ADDRSLT1.OPR)=0))THEN
1398     (***************CHECK THIS C B L O O P V A L*****************)
1399              BEGIN(*FALSE*)
1400                IF SIMSAVB(.CBLOOPVAL.) THEN
1401                 BEGIN(*2*)
1402                  VGG:=SIMSAVE(.CBLOOPVAL.);
1403                  FOR VI:=1 TO MAXCELBLK DO
1404                   BEGIN
1405                    VSTOR:=UPDATYES(.VGG,VI.);
1406                    UPDATYES(.VGG,VI.):=SIMSAVB(.CBLOOPVAL.);
1407                    SIMSAVB(.CBLOOPVAL.):=VSTOR
1408                   END;
1409                  FOR VI:=1 TO MAXCELBLK DO
1410                   BEGIN
1411                    STOR:=UPADDGIV(.VGG,VI.).IC;
1412                    UPADDGIV(.VGG,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1413                    SIMSAVE(.CBLOOPVAL.):=STOR
1414                   END;
1415                  SIMSAVE(.CBLOOPVAL.):=0;
1416                  SIMSAVB(.CBLOOPVAL.):=FALSE
1417                 END;(*2*)
1418               VB:=FALSE;
1419               FOR VI:= 1 TO MAXCELBLK DO
1420                BEGIN(*1*)
1421                 VSTOR:=UPDATYES(.CBLOOPVAL,VI.);
1422                 UPDATYES(.CBLOOPVAL,VI.):=VB;
1423                 VB:=VSTOR
1424                END;(*1*)
1425                 VARX:=0;
```

```
1427 |                  BEGIN
1428 |                   STOR:=UPADDGIV(.CBLOOPVAL,VI.).IC;
1429 |                   UPADDGIV(.CBLOOPVAL,VI.).IC:=VARX;
1430 |                   VARX:=STOR
1431 |                  END;
1432 |              END(*FALSE*)
1433 |             ELSE
1434 |              BEGIN(*1A*)
1435 |              (*GIVING THE ADDRESS VALUE TO THE FETHC UNIT,FOR*)
1436 |                   (*THE INSTRUCTION JUST UPDATED*)
1437 |               IF (VK > CBLOOPVAL) THEN(*BECAUSE FIRST TIME AROUND IT WOULD*)
1438 |                                       (*BE FALSE*************************)
1439 |                BEGIN(*1*)
1440 |                 IF SIMSAVB(.CBLOOPVAL.) THEN
1441 |                  BEGIN(*2*)
1442 |                   FOR VI:=1 TO MAXCELBLK DO
1443 |                    BEGIN
1444 |                     VSTOR:=UPDATYES(.VK,VI.);
1445 |                     UPDATYES(.VK,VI.):=SIMSAVB(.CBLOOPVAL.);
1446 |                     SIMSAVB(.CBLOOPVAL.):=VSTOR
1447 |                    END;
1448 |                   FOR VI:=1 TO MAXCELBLK DO
1449 |                    BEGIN
1450 |                     STOR:=UPADDGIV(.VK,VI.).IC;
1451 |                     UPADDGIV(.VK,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1452 |                     SIMSAVE(.CBLOOPVAL.):=STOR
1453 |                    END;
1454 |                   SIMSAVE(.CBLOOPVAL.):=VK;(*STORING FOR NEXT USE*)
1455 |                   SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1456 |                 END(*2*)
1457 |                ELSE(*IF SIMSAVB(.CBLOOPVAL.) IS FALSE*)
1458 |                 (*JUST STORE FOR THE NEXT VALUE*)
1459 |                   SIMSAVE(.CBLOOPVAL.):=VK;(*STORING FOR NEXT USE*)
1460 |                   SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1461 |           END(*1*)
1462 |          ELSE(*IF VK<= TO CBLOOP VAL*)
1463 |           BEGIN(*3*)
1464 |                 IF SIMSAVB(.CBLOOPVAL.) THEN
1465 |                  BEGIN(*2*)
1466 |                   VGG:=SIMSAVE(.CBLOOPVAL.);
1467 |                   FOR VI:=1 TO MAXCELBLK DO
1468 |                    BEGIN
1469 |                     VSTOR:=UPDATYES(.VGG,VI.);
1470 |                     UPDATYES(.VGG,VI.):=SIMSAVB(.CBLOOPVAL.);
1471 |                     SIMSAVB(.CBLOOPVAL.):=VSTOR
1472 |                    END;
1473 |                   FOR VI:=1 TO MAXCELBLK DO
1474 |                    BEGIN
1475 |                     STOR:=UPADDGIV(.VGG,VI.).IC;
1476 |                     UPADDGIV(.VGG,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1477 |                     SIMSAVE(.CBLOOPVAL.):=STOR
1478 |                    END;
1479 |                   SIMSAVE(.CBLOOPVAL.):=0;
1480 |                   SIMSAVB(.CBLOOPVAL.):=FALSE
1481 |                 END;(*2*)
1482 |               VB:=TRUE;
1483 |               FOR VI:=1 TO MAXCELBLK DO
1484 |                BEGIN
1485 |                 VSTOR:=UPDATYES(.VK,VI.);
1486 |                 UPDATYES(.VK,VI.):=VB;
1487 |                 VB:=VSTOR
1488 |                END;
1489 |               VARX:=DISADDGIV.ADDRSLT1.IC;
1490 |                FOR VI:=1 TO MAXCELBLK DO
```

```
1492 |                  STOR:=UPADDGIV(.VK,VI.).IC;
1493 |                  UPADDGIV(.VK,VI.).IC:=VARX;
1494 |                  VARX:=STOR
1495 |                END;
1496 |            END;(*3*)
1497 |   (*****************NEXT UPDATING THE CELL BLOCK******************)
1498 |   (**************ACTUALLY WRITTING IN THE ARRY*WITH RESULT*****)
1499 |              VOPR:=DISADDGIV.ADDRSLT1.OPR;
1500 |              VARJ:=DISADDGIV.ADDRSLT1.CB;
1501 |              VARK:=DISADDGIV.ADDRSLT1.IC;
1502 |               IF (VOPR=1)THEN
1503 |                CELLBLOCK(.VARJ,VARK.).OPRAND1:=DISADDGIV.RESULT;
1504 |               IF (VOPR=2)THEN
1505 |                CELLBLOCK(.VARJ,VARK.).OPRAND2:=DISADDGIV.RESULT
1506 |             END(*END ELSE*)
1507 |           END;(*END BEGIN*)
1508 |         'D':(**SAME AS 'A' AND 'C' EXCEPT RESULT 2**)
1509 |           BEGIN
1510 |            VK:=DISADDGIV.ADDRSLT2.CB;
1511 |            IF
1512 |            (((DISADDGIV.ADDRSLT2.CB)=0)AND((DISADDGIV.ADDRSLT2.IC)=0)AND
1513 |            ((DISADDGIV.ADDRSLT2.OPR)=0))THEN
1514 |   (***************CHECK THIS C B L O O P V A L*****************)
1515 |              BEGIN(*FALSE*)
1516 |                IF SIMSAVB(.CBLOOPVAL.) THEN
1517 |                 BEGIN(*2*)
1518 |                  VGG:=SIMSAVE(.CBLOOPVAL.);
1519 |                  FOR VI:=1 TO MAXCELBLK DO
1520 |                   BEGIN
1521 |                    VSTOR:=UPDATYES(.VGG,VI.);
1522 |                    UPDATYES(.VGG,VI.):=SIMSAVB(.CBLOOPVAL.);
1523 |                    SIMSAVB(.CBLOOPVAL.):=VSTOR
1524 |                   END;
1525 |                  FOR VI:=1 TO MAXCELBLK DO
1526 |                   BEGIN
1527 |                    STOR:=UPADDGIV(.VGG,VI.).IC;
1528 |                    UPADDGIV(.VGG,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1529 |                    SIMSAVE(.CBLOOPVAL.):=STOR
1530 |                   END;
1531 |                  SIMSAVE(.CBLOOPVAL.):=0;
1532 |                  SIMSAVB(.CBLOOPVAL.):=FALSE
1533 |                END;(*2*)
1534 |              VB:=FALSE;
1535 |              FOR VI:= 1 TO MAXCELBLK DO
1536 |               BEGIN(*1*)
1537 |                VSTOR:=UPDATYES(.CBLOOPVAL,VI.);
1538 |                UPDATYES(.CBLOOPVAL,VI.):=VB;
1539 |                VB:=VSTOR
1540 |               END;(*1*)
1541 |                VARX:=0;
1542 |                FOR VI:=1 TO MAXCELBLK DO
1543 |                 BEGIN
1544 |                  STOR:=UPADDGIV(.CBLOOPVAL,VI.).IC;
1545 |                  UPADDGIV(.CBLOOPVAL,VI.).IC:=VARX;
1546 |                  VARX:=STOR
1547 |                 END;
1548 |             END(*FALSE*)
1549 |           ELSE
1550 |            BEGIN(*1A*)
1551 |           (*GIVING THE ADDRESS VALUE TO THE FETHC UNIT,FOR*)
1552 |                 (*THE INSTRUCTION JUST UPDATED*)
1553 |             IF (VK > CBLOOPVAL) THEN(*BECAUSE FIRST TIME AROUND IT WOULD*)
1554 |                                      (*BE FALSE*************************)
1555 |              BEGIN(*1*)
```

```
1556 |                    FOR VI:=1 TO MAXCELBLK DO
1559 |                     BEGIN
1560 |                      VSTOR:=UPDATYES(.VK,VI.);
1561 |                      UPDATYES(.VK,VI.):=SIMSAVB(.CBLOOPVAL.);
1562 |                      SIMSAVB(.CBLOOPVAL.):=VSTOR
1563 |                     END;
1564 |                    FOR VI:=1 TO MAXCELBLK DO
1565 |                     BEGIN
1566 |                      STOR:=UPADDGIV(.VK,VI.).IC;
1567 |                      UPADDGIV(.VK,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1568 |                      SIMSAVE(.CBLOOPVAL.):=STOR
1569 |                     END;
1570 |                    SIMSAVE(.CBLOOPVAL.):=VK;(*STORING FOR NEXT USE*)
1571 |                    SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1572 |                  END(*2*)
1573 |                ELSE(*IF SIMSAVB(.CBLOOPVAL.) IS FALSE*)
1574 |                 (*JUST STORE FOR THE NEXT VALUE*)
1575 |                    SIMSAVE(.CBLOOPVAL.):=VK;(*STORING FOR NEXT USE*)
1576 |                    SIMSAVB(.CBLOOPVAL.):=TRUE(*STORING FOR NEXT USE*)
1577 |             END(*1*)
1578 |           ELSE(*IF VK<= TO CBLOOP VAL*)
1579 |            BEGIN(*3*)
1580 |                  IF SIMSAVB(.CBLOOPVAL.) THEN
1581 |                   BEGIN(*2*)
1582 |                    VGG:=SIMSAVE(.CBLOOPVAL.);
1583 |                    FOR VI:=1 TO MAXCELBLK DO
1584 |                     BEGIN
1585 |                      VSTOR:=UPDATYES(.VGG,VI.);
1586 |                      UPDATYES(.VGG,VI.):=SIMSAVB(.CBLOOPVAL.);
1587 |                      SIMSAVB(.CBLOOPVAL.):=VSTOR
1588 |                     END;
1589 |                    FOR VI:=1 TO MAXCELBLK DO
1590 |                     BEGIN
1591 |                      STOR:=UPADDGIV(.VGG,VI.).IC;
1592 |                      UPADDGIV(.VGG,VI.).IC:=SIMSAVE(.CBLOOPVAL.);
1593 |                      SIMSAVE(.CBLOOPVAL.):=STOR
1594 |                     END;
1595 |                    SIMSAVE(.CBLOOPVAL.):=0;
1596 |                    SIMSAVB(.CBLOOPVAL.):=FALSE
1597 |                   END;(*2*)
1598 |                VB:=TRUE;
1599 |                FOR VI:=1 TO MAXCELBLK DO
1600 |                 BEGIN
1601 |                  VSTOR:=UPDATYES(.VK,VI.);
1602 |                  UPDATYES(.VK,VI.):=VB;
1603 |                  VB:=VSTOR
1604 |                 END;
1605 |                VARX:=DISADDGIV.ADDRSLT2.IC;
1606 |                 FOR VI:=1 TO MAXCELBLK DO
1607 |                  BEGIN
1608 |                   STOR:=UPADDGIV(.VK,VI.).IC;
1609 |                   UPADDGIV(.VK,VI.).IC:=VARX;
1610 |                   VARX:=STOR
1611 |                  END;
1612 |           END;(*3*)
1613 |   (*****************NEXT UPDATING THE CELL BLOCK*****************)
1614 |   (**************ACTUALLY WRITTING IN THE ARRY*WITH RESULT******)
1615 |                VOPR:=DISADDGIV.ADDRSLT2.OPR;
1616 |                VARJ:=DISADDGIV.ADDRSLT2.CB;
1617 |                VARK:=DISADDGIV.ADDRSLT2.IC;
1618 |                 IF (VOPR=1)THEN
1619 |                  CELLBLOCK(.VARJ,VARK.).OPRAND1:=DISADDGIV.RESULT;
1620 |                 IF (VOPR=2)THEN
```

```
1622 |              END;(*1A*)
1623 |            END;(*END BEGIN*)
1624 |         END(*CASE*)
1625 |    END;(*END PROCEDURE UPDATE*)
1626 |    (*************************************************************************)
1627 |    (*************************************************************************)
1628 |    (*************************************************************************)
1629 |    (**********************M A I N   P R O G R A M**************************)
1630 |    BEGIN(*NSIMARC*)
1631 |                (*INITILIZING ALL INSTRUCTIONS CELLS SUCH THAT OPR1*)
1632 |                (*OPR2 CONTAIN 99(IN HARDWARE IT WOULD BE FF) REPRESENTI*)
1633 |                (*ING BLANK RESULT=99,STAR=A,ADDRSLT1/2=0*)
1634 |                (*NOTE IN ACTUAL HARDWARE IMPLEMENTATION ,THIS WILL*)
1635 |                (*BE TAKEN CARE OF BY MASTER CONTROLLER USING*)
1636 |                (*CELL BLOCK MEM*)
1637 |      FOR I:=1 TO MAXCELBLK DO (*I FOR CELL*)
1638 |        BEGIN
1639 |         FOR J:=1 TO MAXIC DO(*J FOR IC*)
1640 |           BEGIN
1641 |            CELLBLOCK(.I,J.).OPCODE:=NOP;
1642 |            CELLBLOCK(.I,J.).OPRAND1:=99;
1643 |            CELLBLOCK(.I,J.).OPRAND2:=99;
1644 |            CELLBLOCK(.I,J.).RESULT:=99;
1645 |            CELLBLOCK(.I,J.).STAR:='A';
1646 |            CELLBLOCK(.I,J.).ADDRSLT1.CB:=0;
1647 |            CELLBLOCK(.I,J.).ADDRSLT1.IC:=0;
1648 |            CELLBLOCK(.I,J.).ADDRSLT1.OPR:=0;
1649 |            CELLBLOCK(.I,J.).ADDRSLT2.CB:=0;
1650 |            CELLBLOCK(.I,J.).ADDRSLT2.IC:=0;
1651 |            CELLBLOCK(.I,J.).ADDRSLT2.OPR:=0;
1652 |           END
1653 |        END;(*LOOP I,J FINISH*)
1654 |    (*************************************************************************)
1655 |    (**FLUSHING THE PROCESSING ELEMENT PIPELINE TO ZERO*)
1656 |    (**THIS IS THE SIMULATION OF HARDWARE INITILIZATION DONE BY **)
1657 |    (**MASTERCONTROLLER AT THE START OF DFEE****)
1658 |        FOR I:=1 TO MAXCELBLK DO
1659 |         BEGIN
1660 |           FOR J:=1 TO PIPELENGTH DO
1661 |             BEGIN
1662 |              PROCCALC(.I,J.).RESULT:=0;
1663 |              PROCADDRESS(.I,J.).ADDRSLT1.CB:=0;
1664 |              PROCADDRESS(.I,J.).ADDRSLT1.IC:=0;
1665 |              PROCADDRESS(.I,J.).ADDRSLT1.OPR:=0;
1666 |              PROCADDRESS(.I,J.).ADDRSLT2.CB:=0;
1667 |              PROCADDRESS(.I,J.).ADDRSLT2.IC:=0;
1668 |              PROCADDRESS(.I,J.).ADDRSLT2.OPR:=0;
1669 |              PROCADDRESS(.I,J.).STAR:='A'
1670 |             END
1671 |        END;(**LOOP**)
1672 |    (**FOR SIMULATION SOFTWARE WE NEED TO PUT THE UADDGIV TO ZERO**)
1673 |    (**ALSO INITILIZING UPDATYES AND OUTPTRDEY TO FALSE**)
1674 |        FOR I:=1 TO MAXCELBLK DO
1675 |          BEGIN
1676 |           FOR J:=1 TO MAXCELBLK DO
1677 |           BEGIN
1678 |             UPDATYES(.I,J.):=FALSE;
1679 |             OTPTREDY(.I,J.):=FALSE;
1680 |             UPADDGIV(.I,J.).CB:=0;
1681 |             UPADDGIV(.I,J.).IC:=0;
1682 |             UPADDGIV(.I,J.).OPR:=0
1683 |           END
1684 |        END;(**END LOOP**)
1685 |    (******INITILIZING COUNTERS*****)
```

```
1687 |          BEGIN
1688 |            COUNTICNUM(.I.):=1;
1689 |            FETCHREG(.I.):=0;
1690 |            SIMSAVE(.I.):=0;
1691 |            SIMSAVB(.I.):=FALSE
1692 |          END;
1693 |  (*****************************************************************)
1694 |  (*****************************************************************)
1695 |  (******ACTUAL INSTRUCTION CELL ASSIGNMENT **********************)
1696 |  (***T H I S  I S  T H E  O U T P U T  O F  THE ****************)
1697 |  (***************O P T C O M P I L E R*************************)
1698 |  (**CB1:IC:1**)
1699 |  CELLBLOCK(.1,1.).OPCODE:=MUL;
1700 |  CELLBLOCK(.1,1.).OPRAND1:=10;
1701 |  CELLBLOCK(.1,1.).OPRAND2:=1;
1702 |  CELLBLOCK(.1,1.).STAR:='A';
1703 |  CELLBLOCK(.1,1.).ADDRSLT1.CB:=1;
1704 |  CELLBLOCK(.1,1.).ADDRSLT1.IC:=2;
1705 |  CELLBLOCK(.1,1.).ADDRSLT1.OPR:=1;
1706 |  CELLBLOCK(.1,1.).ADDRSLT2.CB:=0;
1707 |  CELLBLOCK(.1,1.).ADDRSLT2.IC:=0;
1708 |  CELLBLOCK(.1,1.).ADDRSLT2.OPR:=0;
1709 |  (*************************************)
1710 |  (****CB1:IC2***)
1711 |  CELLBLOCK(.1,2.).OPCODE:=ADD;
1712 |  CELLBLOCK(.1,2.).OPRAND1:=99;
1713 |  CELLBLOCK(.1,2.).OPRAND2:=10;
1714 |  CELLBLOCK(.1,2.).STAR:='A';
1715 |  CELLBLOCK(.1,2.).ADDRSLT1.CB:=1;
1716 |  CELLBLOCK(.1,2.).ADDRSLT1.IC:=3;
1717 |  CELLBLOCK(.1,2.).ADDRSLT1.OPR:=2;
1718 |  CELLBLOCK(.1,2.).ADDRSLT2.CB:=0;
1719 |  CELLBLOCK(.1,2.).ADDRSLT2.IC:=0;
1720 |  CELLBLOCK(.1,2.).ADDRSLT2.OPR:=0;
1721 |  (*************************************)
1722 |  (****CB1:IC3***)
1723 |  CELLBLOCK(.1,3.).OPCODE:=LOOPI;
1724 |  CELLBLOCK(.1,3.).OPRAND1:=99;
1725 |  CELLBLOCK(.1,3.).OPRAND2:=99;
1726 |  CELLBLOCK(.1,3.).STAR:='A';
1727 |  CELLBLOCK(.1,3.).ADDRSLT1.CB:=1;
1728 |  CELLBLOCK(.1,3.).ADDRSLT1.IC:=4;
1729 |  CELLBLOCK(.1,3.).ADDRSLT1.OPR:=1;
1730 |  CELLBLOCK(.1,3.).ADDRSLT2.CB:=1;
1731 |  CELLBLOCK(.1,3.).ADDRSLT2.IC:=1;
1732 |  CELLBLOCK(.1,3.).ADDRSLT2.OPR:=2;
1733 |  (*************************************)
1734 |  (****CB1:IC4***)
1735 |  CELLBLOCK(.1,4.).OPCODE:=FIN;
1736 |  CELLBLOCK(.1,4.).OPRAND1:=99;
1737 |  CELLBLOCK(.1,4.).OPRAND2:=99;
1738 |  CELLBLOCK(.1,4.).STAR:='A';
1739 |  CELLBLOCK(.1,4.).ADDRSLT1.CB:=0;
1740 |  CELLBLOCK(.1,4.).ADDRSLT1.IC:=0;
1741 |  CELLBLOCK(.1,4.).ADDRSLT1.OPR:=0;
1742 |  CELLBLOCK(.1,4.).ADDRSLT2.CB:=0;
1743 |  CELLBLOCK(.1,4.).ADDRSLT2.IC:=0;
1744 |  CELLBLOCK(.1,4.).ADDRSLT2.OPR:=0;
1745 |  (*************************************)
1746 |  (* C O U N T  S E T  U P  H E R E *)
1747 |  (***CB2:IC1******)
1748 |  CELLBLOCK(.2,1.).OPCODE:=ADD;
1749 |  CELLBLOCK(.2,1.).OPRAND1:=1;
1750 |  CELLBLOCK(.2,1.).OPRAND2:=1;
```

```
1753  |  CELLBLOCK(.2,1.).ADDRSLT1.IC:=3;(****THIS NEEDS TO BE SWAPED DUE TO**)
1754  |  CELLBLOCK(.2,1.).ADDRSLT1.OPR:=2;(*SYNCHRONIZATION WITH CB1****)
1755  |  CELLBLOCK(.2,1.).ADDRSLT2.CB:=2;
1756  |  CELLBLOCK(.2,1.).ADDRSLT2.IC:=2;
1757  |  CELLBLOCK(.2,1.).ADDRSLT2.OPR:=1;
1758  |  (*************************************)
1759  |  (***CB2:IC2******)
1760  |  CELLBLOCK(.2,2.).OPCODE:=COMP;
1761  |  CELLBLOCK(.2,2.).OPRAND1:=99;
1762  |  CELLBLOCK(.2,2.).OPRAND2:=4;(*FVAL*)
1763  |  CELLBLOCK(.2,2.).STAR:='B';
1764  |  CELLBLOCK(.2,2.).ADDRSLT1.CB:=1;
1765  |  CELLBLOCK(.2,2.).ADDRSLT1.IC:=3;
1766  |  CELLBLOCK(.2,2.).ADDRSLT1.OPR:=1;
1767  |  CELLBLOCK(.2,2.).ADDRSLT2.CB:=2;
1768  |  CELLBLOCK(.2,2.).ADDRSLT2.IC:=3;
1769  |  CELLBLOCK(.2,2.).ADDRSLT2.OPR:=1;
1770  |  (*************************************)
1771  |  (*************************************)
1772  |  (***CB2:IC3******)
1773  |  CELLBLOCK(.2,3.).OPCODE:=LOOPI;
1774  |  CELLBLOCK(.2,3.).OPRAND1:=99;
1775  |  CELLBLOCK(.2,3.).OPRAND2:=99;(*FVAL*)
1776  |  CELLBLOCK(.2,3.).STAR:='A';
1777  |  CELLBLOCK(.2,3.).ADDRSLT1.CB:=2;
1778  |  CELLBLOCK(.2,3.).ADDRSLT1.IC:=4;(*DUMP*)
1779  |  CELLBLOCK(.2,3.).ADDRSLT1.OPR:=2;
1780  |  CELLBLOCK(.2,3.).ADDRSLT2.CB:=2;
1781  |  CELLBLOCK(.2,3.).ADDRSLT2.IC:=1;
1782  |  CELLBLOCK(.2,3.).ADDRSLT2.OPR:=1;
1783  |  (*************************************)
1784  |  (**HERE A COUNT WOULD BE USED TO STOP THE PROGRAMME WHILE**)
1785  |  (**IN ACTUAL HARDWARE THIS WOULD BE DONE BY M/C WHICH AFTER**)
1786  |  (**RECEIVING REQUIRED OUTPUTS WILL STOP THE DFEE SYSTEM**)
1787  |     BEGIN(*MAIN PRG*)
1788  |      FLAGOUT:=FALSE;
1789  |      CLOCK:=1;
1790  |  (**STOP WILL BE USED TO STOP THE PROG.SIMULATION AS IF M/C STOPS IT*)
1791  |        WHILE NOT FLAGOUT DO
1792  |          (**IMPORTANT:NO STOPPING CRITERION SPECIFIED***JUST RUN***)
1793  |          BEGIN(*WHILE*)
1794  |           (**CLOCK TELLS US HOW MANUY TIMES THE MACHINE WILL RUN**)
1795  |           (**THROUGH ENCLOSING THE PARRALLEL LOOP**)
1796  |            FOR CBLOOPVAL:=1 TO MAXCELBLK DO
1797  |             BEGIN(*PARALLELISM LOOP*)
1798  |              FETCHUNIT;(*CALL FETCH UNIT*)
1799  |               WRITE('****FETCHUNIT OUTPUT***':8);
1800  |               WRITE('*FETCHUNIT NO':15,CBLOOPVAL:2);
1801  |               WRITE('*FETCHOPRND':13,ORD(FTHADGIV.OPCODE):2);
1802  |               WRITE('*FETCHEDOPR1':12,FTHADGIV.OPRAND1:2);
1803  |               WRITE('*FETCHEDOPR2':12,FTHADGIV.OPRAND2:2);
1804  |               WRITE('*FETCHED RES.BLK':16,FTHADGIV.RESULT:2);
1805  |               WRITE('*FETCHED*':9,FTHADGIV.STAR:3);
1806  |               WRITE('*FETCHED AD1.CB':15,FTHADGIV.ADDRSLT1.CB:2);
1807  |               WRITE('*FETCHED AD1.IC':15,FTHADGIV.ADDRSLT1.IC:2);
1808  |               WRITE('*FETCHED AD1.OPR':16,FTHADGIV.ADDRSLT1.OPR:2);
1809  |               WRITE('*FETCHED AD2.CB':15,FTHADGIV.ADDRSLT2.CB:2);
1810  |               WRITE('*FETCHED AD2.IC':15,FTHADGIV.ADDRSLT2.IC:2);
1811  |               WRITELN('*FETCHED AD2.OPR':16,FTHADGIV.ADDRSLT2.OPR:2);
1812  |               WRITE('***************PROCESSING ELEMENT*****':20);
1813  |  (****C H  E C K THE UPYES AND OTRDY SIG**MAY BE MISTAKE***)
1814  |              PROCELEM;(*CALL PROCELEMT*)
1815  |               WRITE('**RESULT PACKET**':12);
```

```
1817 |                  WRITE('*ADDSTAR':8,RESLTPAC.STAR:4);
1818 |                  WRITE('*ADD1CB':7,RESLTPAC.ADDRSLT1.CB:2);
1819 |                  WRITE('*ADD1IC':7,RESLTPAC.ADDRSLT1.IC:2);
1820 |                  WRITE('*ADD1OPR':8,RESLTPAC.ADDRSLT1.OPR:2);
1821 |                  WRITE('*ADD2CB':7,RESLTPAC.ADDRSLT2.CB:2);
1822 |                  WRITE('*ADD2IC':7,RESLTPAC.ADDRSLT2.IC:2);
1823 |                  WRITELN('*ADD2OPR':8,RESLTPAC.ADDRSLT2.OPR:2);
1824 |   WRITE('************ D I S T.  N E T W O R K*******':20);
1825 |                DISTNET;(*CALLING DISTRIBUTION NETWORK*)
1826 |                 WRITE('*DIS.RES.TRAN':13,DISADDGIV.RESULT:3);
1827 |                 WRITE('*DIS.RES.STAR':13,DISADDGIV.STAR:3);
1828 |                 WRITE('*DIS.ADD1.CB':12,DISADDGIV.ADDRSLT1.CB:2);
1829 |                 WRITE('*DIS.ADD1.IC':12,DISADDGIV.ADDRSLT1.IC:2);
1830 |                 WRITE('*DIS.ADD1.OPR':13,DISADDGIV.ADDRSLT1.OPR:2);
1831 |                 WRITE('DIS.ADD2.CB':12,DISADDGIV.ADDRSLT2.CB:2);
1832 |                 WRITE('DIS.ADD2.IC':12,DISADDGIV.ADDRSLT2.IC:2);
1833 |                 WRITELN('DIS.ADD2.OPR':13,DISADDGIV.ADDRSLT2.OPR:2);
1834 |  WRITELN('COUNT OF CB':10,')':1,CBLOOPVAL:2,COUNTICNUM(.CBLOOPVAL.):2);
1835 | WRITE('************U P D A T E   U N I T******':20);
1836 |               UPDATE;(*CALLING UPDATE UNIT*)
1837 | WRITE('**UPDATEYES':11,CBLOOPVAL:1,')':1,UPDATYES(.CBLOOPVAL,MAXCELBLK.):8);
1838 |                 WRITELN('***MEMDUMP FOLLOWS********':30);
1839 |                  FOR II:=1 TO MAXCELBLK DO
1840 |                   BEGIN
1841 |                    FOR JJ:=1 TO MAXIC DO
1842 |                     BEGIN
1843 |                      WRITE('CELLNO':6,II:2,'ICNO':4,JJ:2);
1844 |                      WRITE('*CELOPCD':8,ORD(CELLBLOCK(.II,JJ.).OPCODE):2
1845 |                      );
1846 |                      WRITE('*CELOPR1':8,CELLBLOCK(.II,JJ.).OPRAND1:2);
1847 |                      WRITE('*CELOPR2':8,CELLBLOCK(.II,JJ.).OPRAND2:2);
1848 |                      WRITE('*CELRES':7,CELLBLOCK(.II,JJ.).RESULT:3);
1849 |                      WRITE('*CELSTAR':8,CELLBLOCK(.II,JJ.).STAR:2);
1850 |                     WRITE('*CELAD1.CB':10,CELLBLOCK(.II,JJ.).ADDRSLT1.CB:2
1851 |                      );
1852 |                     WRITE('*CELAD1.IC':10,CELLBLOCK(.II,JJ.).ADDRSLT1.IC:2
1853 |                      );
1854 |                     WRITE('*CELAD1.OP':10,CELLBLOCK(.II,JJ.).ADDRSLT1.OPR:
1855 |                      2);
1856 |                     WRITE('*CELAD2.CB':10,CELLBLOCK(.II,JJ.).ADDRSLT2.CB:2
1857 |                      );
1858 |                     WRITE('*CELAD2.IC':10,CELLBLOCK(.II,JJ.).ADDRSLT2.IC:2
1859 |                      );
1860 |                     WRITELN('*CELAD2.OP':10,CELLBLOCK(.II,JJ.).ADDRSLT2.OPR:
1861 |                      2);
1862 |                     END
1863 |                    END;(*PRINT LOOP*)
1864 |                  END;(**PARALLELISM LOOP**)
1865 |              WRITELN('************************':10);
1866 |              WRITELN('**CLOCK=':7,CLOCK:3);
1867 |              WRITELN('************************':10);
1868 |              CLOCK:=CLOCK+1
1869 |            END;(*WHILE*)
1870 |          END (*MAIN PRG*)
1871 |        END.(*NSIMARC*)
1872 |  %EOF
Execution begins...
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
```

```
****FETC  *FETCHUNIT NO 1  *FETCHOPRND 3*FETCHEDOPR110*FETCHEDOPR2 1*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 1*FETCHED AD1.IC 2
*FETCHED AD1.OPR 1*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 2
************U P D A **UPDATEYES1)    FALSE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR199*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD2.
OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 1*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR199*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2.
OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 1*FETCHEDOPR1 1*FETCHEDOPR2 1*FETCHED RES.BLK99*FETCHED*  B*FETCHED AD1.CB 2*FETCHED AD1.IC 3
*FETCHED AD1.OPR 2*FETCHED AD2.CB 2*FETCHED AD2.IC 2*FETCHED AD2.OPR 1
****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 2
************U P D A **UPDATEYES2)    FALSE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR199*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD2.
OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 1*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR199*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2.
OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK  1
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
```

```
****FETC  *FETCHUNIT NO 1  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
****************PROC**RESULT PAC*RESULT TO DIS.NT10*ADDSTAR   A*ADD1CB 1*ADD1IC 2*ADD1OPR 1*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN 10*DIS.RES.STAR  A*DIS.ADD1.CB 1*DIS.ADD1.IC 2*DIS.ADD1.OPR 1 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 3
************U P D A **UPDATEYES1)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR110*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD2.
OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 1*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR199*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2.
OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS          0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS          0
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
****************PROC**RESULT PAC*RESULT TO DIS.NT 2*ADDSTAR   B*ADD1CB 2*ADD1IC 3*ADD1OPR 2*ADD2CB 2*ADD2IC 2*ADD2OPR 1
************ D I S T*DIS.RES.TRAN  2*DIS.RES.STAR  B*DIS.ADD1.CB 2*DIS.ADD1.IC 3*DIS.ADD1.OPR 2 DIS.ADD2.CB 2 DIS.ADD2.IC 2 DIS.ADD2
.OPR 1
COUNT OF C) 2 3
************U P D A **UPDATEYES2)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR110*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD2.
OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 1*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 2*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR199*CELOPR2 2*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2.
OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK  2
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS          2
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS          0
```

```
****FETC  *FETCHUNIT NO 1  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
*****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 4
************U P D A **UPDATEYES1)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR110*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD2.
OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 1*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 2*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR199*CELOPR2 2*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2.
OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           2
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           3
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
*****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 4
************U P D A **UPDATEYES2)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR110*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD2.
OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 1*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 2*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR199*CELOPR2 2*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2.
OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK  3
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           2
```

```
****FETC  *FETCHUNIT NO 1  *FETCHOPRND 1*FETCHEDOPR110*FETCHEDOPR210*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 1*FETCHED AD1.IC 3
*FETCHED AD1.OPR 2*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 4
************U P D A **UPDATEYES1)    FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR110*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD2.
OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 1*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 2*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR199*CELOPR2 2*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2.
OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           2
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 2*FETCHEDOPR1 2*FETCHEDOPR2 4*FETCHED RES.BLK99*FETCHED*  B*FETCHED AD1.CB 1*FETCHED AD1.IC 3
*FETCHED AD1.OPR 1*FETCHED AD2.CB 2*FETCHED AD2.IC 3*FETCHED AD2.OPR 1
****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 4
************U P D A **UPDATEYES2)    FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR110*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD2.
OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 1*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 2*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR199*CELOPR2 2*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2.
OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK  4
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
```

```
****FETC  *FETCHUNIT NO 1  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
****************PROC**RESULT PAC*RESULT TO DIS.NT20*ADDSTAR   A*ADD1CB 1*ADD1IC 3*ADD1OPR 2*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN 20*DIS.RES.STAR  A*DIS.ADD1.CB 1*DIS.ADD1.IC 3*DIS.ADD1.OPR 2 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2.
.OPR 0
COUNT OF C) 1 5
************U P D A **UPDATEYES1)    FALSE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR110*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR199*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD2.
OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 1*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 2*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR199*CELOPR2 2*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2.
OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
****************PROC**RESULT PAC*RESULT TO DIS.NT9999*ADDSTAR   B*ADD1CB 1*ADD1IC 3*ADD1OPR 1*ADD2CB 2*ADD2IC 3*ADD2OPR 1
************ D I S T*DIS.RES.TRAN9999*DIS.RES.STAR  B*DIS.ADD1.CB 1*DIS.ADD1.IC 3*DIS.ADD1.OPR 1 DIS.ADD2.CB 2 DIS.ADD2.IC 3 DIS.ADD
2.OPR 1
COUNT OF C) 2 5
************U P D A **UPDATEYES2)    FALSE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR110*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 1*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 2*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 2*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK  5
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS            3
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS            3
```

```
 4*FETCHED AD1.OPR 1*FETCHED AD2.CB 1*FETCHED AD2.IC 1*FETCHED AD2.OPR 2
****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 5
************U P D A **UPDATEYES1)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR110*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 1*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 2*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 2*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS          3
++++D E B U G+++++
++++D E B U G+++++
FETOH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS          0
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 5
************U P D A **UPDATEYES2)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2 1*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR110*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 1*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 2*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 2*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK  6
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS          0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS          3
```

```
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
**********PROC**RESULT PAC*RESULT TO DIS.NT20*ADDSTAR   D*ADD1CB 1*ADD1IC 4*ADD1OPR 1*ADD2CB 1*ADD2IC 1*ADD2OPR 2
********** D I S T*DIS.RES.TRAN 20*DIS.RES.STAR  D*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 1 DIS.ADD2.IC 1 DIS.ADD2
.OPR 2
COUNT OF C) 1 5
**********U P D A **UPDATEYES1)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR110*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 1*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 2*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 2*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           3
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 8*FETCHEDOPR19999*FETCHEDOPR2 2*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 2*FETCHED AD1.IC
 4*FETCHED AD1.OPR 2*FETCHED AD2.CB 2*FETCHED AD2.IC 1*FETCHED AD2.OPR 1
**********PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
********** D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 5
**********U P D A **UPDATEYES2)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR110*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 1*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 2*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 2*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK  7
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           1
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
```

```
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************* D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 5
*************U P D A **UPDATEYES1)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR110*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 1*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 2*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 2*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
****************PROC**RESULT PAC*RESULT TO DIS.NT 2*ADDSTAR   D*ADD1CB 2*ADD1IC 4*ADD1OPR 2*ADD2CB 2*ADD2IC 1*ADD2OPR 1
************* D I S T*DIS.RES.TRAN  2*DIS.RES.STAR  D*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 2 DIS.ADD2.IC 1 DIS.ADD2
.OPR 1
COUNT OF C) 2 5
*************U P D A **UPDATEYES2)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR110*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 2*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 2*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 2*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK  8
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           1
```

```
*FETCHED AD1.OPR 1*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
*****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 5
************U P D A **UPDATEYES1)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR110*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 2*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 2*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 2*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS          1
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS          0
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
*****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 5
************U P D A **UPDATEYES2)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR110*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 2*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 2*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 2*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK  9
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS          0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS          0
```

```
****FETC  *FETCHUNIT NO 1  *FETCHOPRND 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
*****************PROC**RESULT PAC*RESULT TO DIS.NT200*ADDSTAR   A*ADD1CB 1*ADD1IC 2*ADD1OPR 1*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN200*DIS.RES.STAR  A*DIS.ADD1.CB 1*DIS.ADD1.IC 2*DIS.ADD1.OPR 1 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 5
************U P D A **UPDATEYES1)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 2*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 2*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 2*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           1
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 1*FETCHEDOPR1 2*FETCHEDOPR2 1*FETCHED RES.BLK99*FETCHED*  B*FETCHED AD1.CB 2*FETCHED AD1.IC 3
*FETCHED AD1.OPR 2*FETCHED AD2.CB 2*FETCHED AD2.IC 2*FETCHED AD2.OPR 1
*****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 5
************U P D A **UPDATEYES2)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 2*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 2*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 2*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK 10
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           2
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
```

```
****FETC  *FETCHUNIT NO 1  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 5
************U P D A **UPDATEYES1)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 2*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 2*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 2*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
****************PROC**RESULT PAC*RESULT TO DIS.NT 3*ADDSTAR   B*ADD1CB 2*ADD1IC 3*ADD1OPR 2*ADD2CB 2*ADD2IC 2*ADD2OPR 1
************ D I S T*DIS.RES.TRAN  3*DIS.RES.STAR  B*DIS.ADD1.CB 2*DIS.ADD1.IC 3*DIS.ADD1.OPR 2 DIS.ADD2.CB 2 DIS.ADD2.IC 2 DIS.ADD2
.OPR 1
COUNT OF C) 2 5
************U P D A **UPDATEYES2)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 2*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 3*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 3*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK 11
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           2
```

```
3*FETCHED AD1.OPR 2*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 5
************U P D A **UPDATEYES1)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 2*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 3*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 3*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           2
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           3
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 8*FETCHEDOPR19999*FETCHEDOPR2 3*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 2*FETCHED AD1.IC
 4*FETCHED AD1.OPR 2*FETCHED AD2.CB 2*FETCHED AD2.IC 1*FETCHED AD2.OPR 1
****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 5
************U P D A **UPDATEYES2)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 2*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 3*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 3*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK 12
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
```

```
****FETC  *FETCHUNIT NO 1  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
*****************PROC**RESULT PAC*RESULT TO DIS.NT210*ADDSTAR   A*ADD1CB 1*ADD1IC 3*ADD1OPR 2*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN210*DIS.RES.STAR  A*DIS.ADD1.CB 1*DIS.ADD1.IC 3*DIS.ADD1.OPR 2 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 5
*************U P D A **UPDATEYES1)    FALSE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 2*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 3*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 3*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS            2
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 7*FETCHEDOPR1 3*FETCHEDOPR2 4*FETCHED RES.BLK99*FETCHED*  B*FETCHED AD1.CB 1*FETCHED AD1.IC 3
*FETCHED AD1.OPR 1*FETCHED AD2.CB 2*FETCHED AD2.IC 3*FETCHED AD2.OPR 1
*****************PROC**RESULT PAC*RESULT TO DIS.NT 3*ADDSTAR   D*ADD1CB 2*ADD1IC 4*ADD1OPR 2*ADD2CB 2*ADD2IC 1*ADD2OPR 1
************ D I S T*DIS.RES.TRAN  3*DIS.RES.STAR  D*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 2 DIS.ADD2.IC 1 DIS.ADD2
.OPR 1
COUNT OF C) 2 5
*************U P D A **UPDATEYES2)    FALSE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 3*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 3*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK 13
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS            3
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
```

```
**********PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
********** D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 5
**********U P D A **UPDATEYES1)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 3*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 3*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           1
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
**********PROC**RESULT PAC*RESULT TO DIS.NT9999*ADDSTAR   B*ADD1CB 1*ADD1IC 3*ADD1OPR 1*ADD2CB 2*ADD2IC 3*ADD2OPR 1
********** D I S T*DIS.RES.TRAN9999*DIS.RES.STAR  B*DIS.ADD1.CB 1*DIS.ADD1.IC 3*DIS.ADD1.OPR 1 DIS.ADD2.CB 2 DIS.ADD2.IC 3 DIS.ADD
2.OPR 1
COUNT OF C) 2 5
**********U P D A **UPDATEYES2)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 3*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 3*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK 14
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           3
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
```

```
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************* D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 5
*************U P D A **UPDATEYES1)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 3*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 3*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           3
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           1
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 1*FETCHEDOPR1 3*FETCHEDOPR2 1*FETCHED RES.BLK99*FETCHED*  B*FETCHED AD1.CB 2*FETCHED AD1.IC 3
*FETCHED AD1.OPR 2*FETCHED AD2.CB 2*FETCHED AD2.IC 2*FETCHED AD2.OPR 1
****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************* D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 5
*************U P D A **UPDATEYES2)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 3*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 3*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK 15
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           3
```

```
C 4*FETCHED AD1.OPR 1*FETCHED AD2.CB 1*FETCHED AD2.IC 1*FETCHED AD2.OPR 2
**************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 5
************U P D A **UPDATEYES1)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 3*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 3*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS          0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS          3
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 8*FETCHEDOPR19999*FETCHEDOPR2 3*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 2*FETCHED AD1.IC
 4*FETCHED AD1.OPR 2*FETCHED AD2.CB 2*FETCHED AD2.IC 1*FETCHED AD2.OPR 1
**************PROC**RESULT PAC*RESULT TO DIS.NT 4*ADDSTAR   B*ADD1CB 2*ADD1IC 3*ADD1OPR 2*ADD2CB 2*ADD2IC 2*ADD2OPR 1
************ D I S T*DIS.RES.TRAN  4*DIS.RES.STAR  B*DIS.ADD1.CB 2*DIS.ADD1.IC 3*DIS.ADD1.OPR 2 DIS.ADD2.CB 2 DIS.ADD2.IC 2 DIS.ADD2
.OPR 1
COUNT OF C) 2 5
************U P D A **UPDATEYES2)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR220*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK 16
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS          0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS          0
```

```
****FETC  *FETCHUNIT NO 1  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
*****************PROC**RESULT PAC*RESULT TO DIS.NT210*ADDSTAR   D*ADD1CB 1*ADD1IC 4*ADD1OPR 1*ADD2CB 1*ADD2IC 1*ADD2OPR 2
************ D I S T*DIS.RES.TRAN210*DIS.RES.STAR  D*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 1 DIS.ADD2.IC 1 DIS.ADD2
.OPR 2
COUNT OF C) 1 5
*************U P D A **UPDATEYES1)    FALSE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           2
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           3
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
*****************PROC**RESULT PAC*RESULT TO DIS.NT 3*ADDSTAR   D*ADD1CB 2*ADD1IC 4*ADD1OPR 2*ADD2CB 2*ADD2IC 1*ADD2OPR 1
************ D I S T*DIS.RES.TRAN  3*DIS.RES.STAR  D*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 2 DIS.ADD2.IC 1 DIS.ADD2
.OPR 1
COUNT OF C) 2 5
*************U P D A **UPDATEYES2)    TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK 17
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           1
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
```

```
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
**************PROC**RESULT PAC*RESULT TO DIS.NI 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 5
************U P D A **UPDATEYES1)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS          1
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS          2
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 7*FETCHEDOPR1 4*FETCHEDOPR2 4*FETCHED RES.BLK99*FETCHED*  B*FETCHED AD1.CB 1*FETCHED AD1.IC 3
*FETCHED AD1.OPR 1*FETCHED AD2.CB 2*FETCHED AD2.IC 3*FETCHED AD2.OPR 1
**************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 5
************U P D A **UPDATEYES2)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK 18
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS          0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS          1
```

```
2*FETCHED AD1.OPR 1*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
**************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 5
************U P D A **UPDATEYES1)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR19999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD
2.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           1
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 1*FETCHEDOPR1 3*FETCHEDOPR2 1*FETCHED RES.BLK99*FETCHED*  B*FETCHED AD1.CB 2*FETCHED AD1.IC 3
*FETCHED AD1.OPR 2*FETCHED AD2.CB 2*FETCHED AD2.IC 2*FETCHED AD2.OPR 1
**************PROC**RESULT PAC*RESULT TO DIS.NT999*ADDSTAR   B*ADD1CB 1*ADD1IC 3*ADD1OPR 1*ADD2CB 2*ADD2IC 3*ADD2OPR 1
************ D I S T*DIS.RES.TRAN999*DIS.RES.STAR  B*DIS.ADD1.CB 1*DIS.ADD1.IC 3*DIS.ADD1.OPR 1 DIS.ADD2.CB 2 DIS.ADD2.IC 3 DIS.ADD2
.OPR 1
COUNT OF C) 2 5
************U P D A **UPDATEYES2)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR1200*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2
.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK 19
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           3
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
```

```
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
****************PROC**RESULT PAC*RESULT TO DIS.NT2100*ADDSTAR   A*ADD1CB 1*ADD1IC 2*ADD1OPR 1*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN2100*DIS.RES.STAR  A*DIS.ADD1.CB 1*DIS.ADD1.IC 2*DIS.ADD1.OPR 1 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD
2.OPR 0
COUNT OF C) 1 5
************U P D A **UPDATEYES1)    TRUE      ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR12100*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2
.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS            3
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
****************PROC**RESULT PAC*RESULT TO DIS.NT 4*ADDSTAR   B*ADD1CB 2*ADD1IC 3*ADD1OPR 2*ADD2CB 2*ADD2IC 2*ADD2OPR 1
************ D I S T*DIS.RES.TRAN  4*DIS.RES.STAR  B*DIS.ADD1.CB 2*DIS.ADD1.IC 3*DIS.ADD1.OPR 2 DIS.ADD2.CB 2 DIS.ADD2.IC 2 DIS.ADD2
.OPR 1
COUNT OF C) 2 5
************U P D A **UPDATEYES2)    TRUE      ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR12100*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2
.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK 20
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS            2
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS            3
```

```
 4*FETCHED AD1.OPR 1*FFTCHFD AD2.CB 1*FETCHED AD2.IC 1*FETCHED AD2.OPR 2
*****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 5
************U P D A **UPDATEYES1)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR12100*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2
.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           2
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           3
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 8*FETCHEDOPR1999*FETCHEDOPR2 4*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 2*FETCHED AD1.IC
4*FETCHED AD1.OPR 2*FETCHED AD2.CB 2*FETCHED AD2.IC 1*FETCHED AD2.OPR 1
*****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 5
************U P D A **UPDATEYES2)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR12100*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2
.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK 21
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           2
```

```
**********PROC**RESULT PAC*RESULT TO DIS.NT210*ADDSTAR   C*ADD1CB 1*ADD1IC 4*ADD1OPR 1*ADD2CB 1*ADD2IC 1*ADD2OPR 2
********** D I S T*DIS.RES.TRAN210*DIS.RES.STAR  C*DIS.ADD1.CB 1*DIS.ADD1.IC 4*DIS.ADD1.OPR 1 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 5
**********U P D A **UPDATEYES1)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR12100*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR1210*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2
.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           2
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 7*FETCHEDOPR1 4*FETCHEDOPR2 4*FETCHED RES.BLK99*FETCHED*  B*FETCHED AD1.CB 1*FETCHED AD1.IC 3
*FETCHED AD1.OPR 1*FETCHED AD2.CB 2*FETCHED AD2.IC 3*FETCHED AD2.OPR 1
**********PROC**RESULT PAC*RESULT TO DIS.NT 4*ADDSTAR   C*ADD1CB 2*ADD1IC 4*ADD1OPR 2*ADD2CB 2*ADD2IC 1*ADD2OPR 1
********** D I S T*DIS.RES.TRAN  4*DIS.RES.STAR  C*DIS.ADD1.CB 2*DIS.ADD1.IC 4*DIS.ADD1.OPR 2 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 5
**********U P D A **UPDATEYES2)   FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR12100*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELAD
2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR1210*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2
.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK 22
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           4
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
```

```
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
*****************PROC**RESULT PAC*RESULT TO DIS.NI2110*ADDSTAR   A*ADD1CB 1*ADD1IC 3*ADD1OPR 2*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************* D I S T*DIS.RES.TRAN2110*DIS.RES.STAR  A*DIS.ADD1.CB 1*DIS.ADD1.IC 3*DIS.ADD1.OPR 2 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD
2.OPR 0
COUNT OF C) 1 5
*************U P D A **UPDATEYES1)    TRUE      ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR12100*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR1999*CELOPR22110*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR1210*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2
.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           4
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS           0
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
*****************PROC**RESULT PAC*RESULT TO DIS.NT999*ADDSTAR   B*ADD1CB 1*ADD1IC 3*ADD1OPR 1*ADD2CB 2*ADD2IC 3*ADD2OPR 1
************* D I S T*DIS.RES.TRAN999*DIS.RES.STAR  B*DIS.ADD1.CB 1*DIS.ADD1.IC 3*DIS.ADD1.OPR 1 DIS.ADD2.CB 2 DIS.ADD2.IC 3 DIS.ADD2
.OPR 1
COUNT OF C) 2 5
*************U P D A **UPDATEYES2)    TRUE      ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR12100*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR1999*CELOPR22110*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR1210*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2
.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK 23
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           3
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS           3
```

```
C 4*FETCHED AD1.OPR 1*FETCHED AD2.CB 1*FETCHED AD2.IC 1*FETCHED AD2.OPR 2
****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 5
************U P D A **UPDATEYES1)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR12100*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR1999*CELOPR22110*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR1210*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2
.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS          3
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS          4
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 5
************U P D A **UPDATEYES2)   TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR12100*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR1999*CELOPR22110*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR1210*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2
.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK 24
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS          0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS          3
```

```
****FETC  *FETCHUNIT NO 1  *FETCHOPRND 8*FETCHEDOPR1 8*FETCHEDOPR2 8*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
****************PROC**RESULT PAC*RESULT TO DIS.NT2110*ADDSTAR   C*ADD1CB 1*ADD1IC 4*ADD1OPR 1*ADD2CB 1*ADD2IC 1*ADD2OPR 2
************* D I S T*DIS.RES.TRAN2110*DIS.RES.STAR  C*DIS.ADD1.CB 1*DIS.ADD1.IC 4*DIS.ADD1.OPR 1 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD
2.OPR 0
COUNT OF C) 1 5
*************U P D A **UPDATEYES1)    FALSE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR12100*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR1999*CELOPR22110*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR12110*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2
.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS            3
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 8*FETCHEDOPR1999*FETCHEDOPR2 4*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 2*FETCHED AD1.IC
4*FETCHED AD1.OPR 2*FETCHED AD2.CB 2*FETCHED AD2.IC 1*FETCHED AD2.OPR 1
****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************* D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 5
*************U P D A **UPDATEYES2)    FALSE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR12100*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR1999*CELOPR22110*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR12110*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2
.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK 25
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS            4
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
```

```
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
*****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 1 5
************U P D A **UPDATEYES1)    TRUE     ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR12100*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR1999*CELOPR22110*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR12110*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2
.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
****FETC  *FETCHUNIT NO 2  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
*****************PROC**RESULT PAC*RESULT TO DIS.NT 4*ADDSTAR   C*ADD1CB 2*ADD1IC 4*ADD1OPR 2*ADD2CB 2*ADD2IC 1*ADD2OPR 1
************ D I S T*DIS.RES.TRAN  4*DIS.RES.STAR  C*DIS.ADD1.CB 2*DIS.ADD1.IC 4*DIS.ADD1.OPR 2 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
COUNT OF C) 2 5
************U P D A **UPDATEYES2)    FALSE    ***MEMDUMP FOLLOWS********
CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
CELLNO 1ICNO 2*CELOPCD 1*CELOPR12100*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
CELLNO 1ICNO 3*CELOPCD 8*CELOPR1999*CELOPR22110*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
CELLNO 1ICNO 4*CELOPCD 9*CELOPR12110*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
CELLNO 2ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2
.OP 1
CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
**********
**CLOCK 26
**********
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESFALSE
FET UNIT PIPLINE UPADDRESS            0
++++D E B U G+++++
++++D E B U G+++++
FETCH UNIT PIPLINE UP YESTRUE
FET UNIT PIPLINE UPADDRESS            4
```

```
 FIN OPCODE FOUND F  1
   THE RESULT IS2110
 ************************************************
 ****FETC  *FETCHUNIT NO 1  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
 ****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
 ************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
 COUNT OF C) 1 5
 ************U P D A **UPDATEYES1)    FALSE     ***MEMDUMP FOLLOWS********
 CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
 CELLNO 1ICNO 2*CELOPCD 1*CELOPR12100*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
 CELLNO 1ICNO 3*CELOPCD 8*CELOPR1999*CELOPR22110*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
 CELLNO 1ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
 CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
 CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
 CELLNO 2ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2
.OP 1
 CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
 ++++D E B U G+++++
 FETCH UNIT PIPLINE UP YESTRUE
 FET UNIT PIPLINE UPADDRESS           4
 ++++D E B U G+++++
 ++++D E B U G+++++
 FETCH UNIT PIPLINE UP YESFALSE
 FET UNIT PIPLINE UPADDRESS           0
 ++++D E B U G+++++
 ****FETC  *FETCHUNIT NO 2  *FETCHOPRND 0*FETCHEDOPR1 0*FETCHEDOPR2 0*FETCHED RES.BLK99*FETCHED*  A*FETCHED AD1.CB 0*FETCHED AD1.IC 0
*FETCHED AD1.OPR 0*FETCHED AD2.CB 0*FETCHED AD2.IC 0*FETCHED AD2.OPR 0
 ****************PROC**RESULT PAC*RESULT TO DIS.NT 0*ADDSTAR   A*ADD1CB 0*ADD1IC 0*ADD1OPR 0*ADD2CB 0*ADD2IC 0*ADD2OPR 0
 ************ D I S T*DIS.RES.TRAN  0*DIS.RES.STAR  A*DIS.ADD1.CB 0*DIS.ADD1.IC 0*DIS.ADD1.OPR 0 DIS.ADD2.CB 0 DIS.ADD2.IC 0 DIS.ADD2
.OPR 0
 COUNT OF C) 2 5
 ************U P D A **UPDATEYES2)    TRUE      ***MEMDUMP FOLLOWS********
 CELLNO 1ICNO 1*CELOPCD 3*CELOPR110*CELOPR2210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 2*CELAD1.OP 1*CELAD2.CB 0*CELAD2.IC 0*CELAD2
.OP 0
 CELLNO 1ICNO 2*CELOPCD 1*CELOPR12100*CELOPR210*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 0*CELAD2.IC 0*CELAD
2.OP 0
 CELLNO 1ICNO 3*CELOPCD 8*CELOPR1999*CELOPR22110*CELRES 99*CELSTAR A*CELAD1.CB 1*CELAD1.IC 4*CELAD1.OP 1*CELAD2.CB 1*CELAD2.IC 1*CELA
D2.OP 2
 CELLNO 1ICNO 4*CELOPCD 0*CELOPR199*CELOPR299*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
 CELLNO 2ICNO 1*CELOPCD 1*CELOPR1 3*CELOPR2 1*CELRES 99*CELSTAR B*CELAD1.CB 2*CELAD1.IC 3*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 2*CELAD2.
OP 1
 CELLNO 2ICNO 2*CELOPCD 7*CELOPR1 4*CELOPR2 4*CELRES 99*CELSTAR B*CELAD1.CB 1*CELAD1.IC 3*CELAD1.OP 1*CELAD2.CB 2*CELAD2.IC 3*CELAD2.
OP 1
 CELLNO 2ICNO 3*CELOPCD 8*CELOPR1999*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 2*CELAD1.IC 4*CELAD1.OP 2*CELAD2.CB 2*CELAD2.IC 1*CELAD2
.OP 1
 CELLNO 2ICNO 4*CELOPCD 0*CELOPR199*CELOPR2 4*CELRES 99*CELSTAR A*CELAD1.CB 0*CELAD1.IC 0*CELAD1.OP 0*CELAD2.CB 0*CELAD2.IC 0*CELAD2.
OP 0
 **********
 **CLOCK 27
 **********

 ...execution ends
 File 'LOOPINS': 1872 lines; no diagnostics
```

```
17776 statements executed
169112 bytes of memory requested during compilation
5792 bytes returned before execution
10296 bytes requested during execution
```

APPENDIX C

PROGRAM AND OUTPUT LISTING OF:
1)ADD/SUBTRACT OPERATION WITH LOGIC II.
2)MULTIPLY OPERATION.
3)DIVIDE OPERATION.
4)SQUARE OPERATION.

S I M U L A T I O N P R O G R A M F O R

A D D/ S U B T R A C T O P E R A T I O N W I T H L O G I C II

I N C L U D E D

```
**************************************************************
WATERLOO PASCAL -- ( CMS, VERSION 3.1 ):  University of Petroleum & Minerals, Dhahran

  1 |  (* $ 50000 *)
  2 |  PROGRAM PROCELEM(INPUT,OUTPUT);
  3 |  (* SIMULATION PROGRAM FOR THE PE INCLUDING THE MODIFICATION OF******)
  4 |  (* ADDITION/SUBSTRACTION. THE CARRY GENERATED IS GIVEN TO THE OTHER*)
  5 |  (* THIS PROGRAM ONLY TESTS THE SIMULATION OF ADD SUBTRACT     ******)
  6 |  (* MODIFICATION,PLEASE NOTE:THIS IS A FOUR BIT SIMULATOR*)
  7 |  (* EXTENSION CAN BE DONE AS REQUIRED*****)
  8 |  CONST
  9 |         XGLB=TRUE;(*X IS THE CONTROL BIT FOR MULT/DIV,SQRT/SQ*)
 10 |         (***Y WILL DISABLE XGLB WHEN IN ADDTION***)
 11 |         Y=TRUE;(* Y IS THE CONTROL BIT FOR ADD/SUB*)
 12 |                (*Y=1 ADD/SUB,Y=0 OTHER*)
 13 |                (*THIS IS A GLOBAL CONTROL VARIABLE*)
 14 |         AS=FALSE;(*THE SIGN BIT FOR A NUMBER*)
 15 |         BS=FALSE;(*THE SIGN BIT FOR B NUMBER*)
 16 |         M=FALSE;(*M=0 FOR ADD, 1 SUB*)
 17 |         ENX=FALSE;(*0ADD/SUB,1 OTHER*)
 18 |        LL=11;(*NP:LL SHOULD BE ODD,INCREASING THE ARRAY BIT WISE COLUMN*)
 19 |        LLL=9;(*TWO LESS THAN LL*)
 20 |         MM=5;(*INCREASING THE ARRAY ROW WISE.I.EPBITS*)
 21 |  TYPE
 22 |      SUB1= 1..LL;(*NOTE:THE EXTENSION OF PE WOULD BE DONE BY SUB1,2*)
 23 |      SUB2= 1..MM;
 24 |      SUB3= 1..LLL;
 25 |      MATR1=ARRAY(.SUB1.)OF BOOLEAN;
 26 |      MATR2=ARRAY(.SUB2.)OF BOOLEAN;
 27 |      MTRAN=ARRAY(.SUB1.)OF INTEGER;
 28 |      MTRAN1=ARRAY(.SUB2.)OF INTEGER;
 29 |      MSAVE=ARRAY(.SUB3.)OF BOOLEAN;
 30 |  VAR
 31 |      A,B,C,C1,Z:MATR1;    (*C1 IS THE CARRY*)
 32 |      F,C0,P:MATR2; (*NOTE:C1(J) IS CARRYOUT WHILE C1(J+1) IS CARRY-IN*)
 33 |                    (*FOR ADDITION/SUBSTRACTION*)
 34 |      COUNT1,COUNT,COLINDX,J,TR,INDX1:INTEGER;
 35 |      A1,B1,C11,Z1:MTRAN;
 36 |      P1:MTRAN1;
 37 |      SAV1,SAV2,SAV3:MSAVE;
 38 |      TRAN1,X,SUMSIGN,OVF,QAT,CB,CAROT:BOOLEAN;
 39 |  (*****************************************************************)
 40 |  (*****************************************************************)
 41 |  (*******************FUNCTION DEFINATIONS**************************)
 42 |  FUNCTION EXOR(LL1,LL2:BOOLEAN):BOOLEAN;
 43 |    BEGIN
 44 |      EXOR:= (LL1 AND (NOT LL2)) OR ( LL2 AND (NOT LL1))
 45 |    END;
 46 |  (*****************************************************************)
 47 |  FUNCTION COMPS(G1,G2,G3,G4,G5,G6:BOOLEAN):BOOLEAN;
 48 |    VAR
 49 |       OPER:BOOLEAN;
 50 |   BEGIN
 51 |    OPER:=EXOR(G2,G5);
```

```
54 |     COMPS:=(OPER AND G4 AND (NOT G6))OR(G1 AND (NOT G4) AND (NOT G6))
55 |   END;(*COMPS FINISHED*)
56 |  (*************************************************************************)
57 |  FUNCTION COMPC0 (H1,H2,H3,H4,H5:BOOLEAN):BOOLEAN;
58 |    VAR
59 |       OPER:BOOLEAN;
60 |   BEGIN
61 |    OPER:= EXOR(H2,H4);
62 |    OPER:= OPER AND (H1 OR H3);
63 |    OPER:=OPER AND (NOT H5);
64 |    OPER:= OPER OR ( H1 AND H3 AND (NOT H5));
65 |    COMPC0:=OPER OR (H3 AND H5)
66 |   END;(*COMC0 FINISH *)
67 |  (*************************************************************************)
68 |  FUNCTION COMPD(P1,P2,P3:BOOLEAN):BOOLEAN;
69 |    VAR
70 |       OPER:BOOLEAN;
71 |   BEGIN
72 |    OPER:=(P1 AND P2) OR (P2 AND P3);
73 |    COMPD:=(OPER AND (NOT Y))OR (X AND Y)(*NOTICE X,Y GLOBAL CONSTANT*)
74 |   END;(*COMPD FINISH*)
75 |  (*************************************************************************)
76 |  FUNCTION COMPE (Q1,Q2,Q3:BOOLEAN):BOOLEAN;
77 |    VAR
78 |       OPER:BOOLEAN;
79 |   BEGIN
80 |    OPER:= Q1 OR ( Q2 AND Q3);
81 |    COMPE:=OPER AND (NOT Y)
82 |   END;(*COMPE FINISH *)
83 |  (*************************************************************************)
84 |  FUNCTION COMPF (R1,R2,R3:BOOLEAN):BOOLEAN;
85 |    VAR
86 |       OPER:BOOLEAN;
87 |   BEGIN
88 |    OPER := (R1 AND R3) OR ( R2 AND (NOT R3));
89 |    COMPF:=OPER OR Y
90 |   END;(*COMPF FINISHED*)
91 |  (*************************************************************************)
92 |  FUNCTION CONV (S1:BOOLEAN):INTEGER;
93 |   BEGIN
94 |    IF S1 = TRUE
95 |            THEN CONV:=1
96 |            ELSE
97 |             CONV:=0
98 |   END;(*CONV FINISHED*)
99 |  (************FUNCTION TO CONVERT INT TO BINARY********)
100|  FUNCTION INTOBIN(VALUE:INTEGER):BOOLEAN;
101|   BEGIN
102|     IF VALUE = 1
103|       THEN INTOBIN:=TRUE
104|       ELSE
105|       IF VALUE =0
106|        THEN INTOBIN:=FALSE
107|   END;(*INTOBIN FINISH*)
108|  (*********MAIN PROGRAM*********)
109|  (*************************************************************************)
110|  (*************************************************************************)
111|  (**********************M A I N   P R O G R A M**************************)
112|  (*READING THE INPUT VALUES WHICH SHOULD BE INPUTED LINE BY LINE********)
113|  (*EITHER 1 OR 0,ONE INPUT ONE LINE*************************************)
114|  (********NOTE:MSB=A(1) AND LOWER ***************************)
115|  (***********************READING LOOP FOR INITIAL VALUE OF A,B,C,P*******)
116|  BEGIN(****MAIN PROGRAM BEGIN LOOP***************)
```

```
118 |  (*********************GENERATING THE VALUE OF X FROM ,AS,BS,Y,XGLB**)
119 |    X:=EXOR(BS,M);
120 |    X:=EXOR(AS,X);
121 |    X:=(X AND Y);
122 |    X:=(X)OR((XGLB)AND(NOT Y));
123 |  (*THIS IS THE INPUT SECTION*************************************)
124 |      A1(.1.):=0;(*THIS BIT NOT BE USED IN DIVISION ALWAYS SET TO 0,SEE FIG*)
125 |      A1(.2.):=0;
126 |      A1(.3.):=0;(**)(***CARRY OUT FOR 4-BIT SIMULATOR**)
127 |      A1(.4.):=0;
128 |      A1(.5.):=0;(**)
129 |      A1(.6.):=0;
130 |      A1(.7.):=1;(**)
131 |      A1(.8.):=0;
132 |      A1(.9.):=1;(**)
133 |      A1(.10.):=0;
134 |      A1(.11.):=1;(**)
135 |  (*********************************************************************)
136 |  (******CONTROL BIT Z  FOR EACH CELL .THIS WILL BE PROPAGATED THROUGH*)
137 |  (****UNCHANGED*****************************************************)
138 |      Z1(.1.):=1;
139 |      Z1(.2.):=1;
140 |      Z1(.3.):=0;
141 |      Z1(.4.):=1;
142 |      Z1(.5.):=0;
143 |      Z1(.6.):=1;
144 |      Z1(.7.):=0;
145 |      Z1(.8.):=1;
146 |      Z1(.9.):=0;
147 |      Z1(.10.):=1;
148 |      Z1(.11.):=0;
149 |  (*****************************************************************)
150 |      B1(.1.):=CONV(X);(*THESE BITS ARE SET TO X*)
151 |      B1(.2.):=CONV(X);(**ALL THOSE BITS WHICH ARE NOT USED*)
152 |      B1(.3.):=CONV(X);(*INCLUDING THE CARRY BIT*)
153 |      B1(.4.):=0;
154 |      B1(.5.):=0;(**)
155 |      B1(.6.):=0;
156 |      B1(.7.):=0;(**)
157 |      B1(.8.):=0;
158 |      B1(.9.):=1;(**)
159 |      B1(.10.):=0;
160 |      B1(.11.):=1;(**)
161 |  (*********************************************************************)
162 |      C11(.1.):=0;
163 |      C11(.2.):=0;
164 |      C11(.3.):=0;
165 |      C11(.4.):=0;
166 |      C11(.5.):=0;
167 |      C11(.6.):=0;
168 |      C11(.7.):=0;
169 |      C11(.8.):=0;
170 |      C11(.9.):=0;
171 |      C11(.10.):=0;
172 |      C11(.11.):=0;
173 |  (*********************************************************************)
174 |      P1(.1.):=0;
175 |      P1(.2.):=0;
176 |      P1(.3.):=0;
177 |      P1(.4.):=0;
178 |      P1(.5.):=0;
179 |  (*********************************************************************)
180 |    FOR J:=1 TO LL DO
181 |     BEGIN(*READING THE INITIAL VALUES OF A*)
```

```
      END;
(****NOTICE: THE A,B,C ARE ALL 11 BITS NOW************************)
(*B:C:4,6,8,10 ARE DUMMY*)
(*B:C:5,3,7,9,11 ARE OVERWRITTEN WITH THE LAST CELL COMPUTATION*)
  FOR J:=1 TO LL DO
   BEGIN(*READING THE INITIAL VALUE OF B*)
    WRITELN('B(':2,J:1,')':2,B1(.J.):1);
    B(.J.):=INTOBIN(B1(.J.))
   END;
(*************************************************************************)
  FOR J:=1 TO LL DO
   BEGIN(*READING THE INITIAL VALUE OF C*)
    WRITELN('C(':2,J:1,')':2,C11(.J.):1);
    C(.J.):=INTOBIN(C11(.J.))
   END;
(*************************************************************************)
  FOR J:=1 TO LL DO
   BEGIN(*READING THE INITIAL VALUE OF Z*)
    WRITELN('Z(':2,J:1,')':2,Z1(.J.):1);
    Z(.J.):=INTOBIN(Z1(.J.))
   END;
(*************************************************************************)
  FOR J:=1 TO MM DO
   BEGIN(*READING THE INITIAL VALUE OF P*)
    WRITELN('P(':2,J:1,')':1,P1(.J.):1);
    P(.J.):=INTOBIN(P1(.J.))
   END;
(************************INITIAL READING VALUE FINISHED**************)
(************THE MODIFICATION OF CARRRY OUT*************)
  CB:=X AND ENX;
(**NOTE THE OCCURANCE OF CB ONLY IN THE LAST CELLS*)
(************THE MODIFICATION OF CARRRY OUT*************)
   BEGIN(*INITILIZATION*)
      COUNT1:=1;
      COUNT:=1;
      INDX1:=0;
      COLINDX:=3;
      WHILE COUNT1<= (MM + 1) DO
         BEGIN(*WHILE COUNT*)
              C1(.COLINDX.):= COMPCO(A(.COLINDX.),B(.COLINDX.),CB,X,
              Z(.COLINDX.));
                FOR J:=(COLINDX -1) DOWNTO 2 DO
                   BEGIN
                    C1(.J.):=COMPCO(A(.J.),B(.(J-1).),C1(.J+1.),X,
                    Z(.J.));
                   END;
                C1(.1.):=COMPCO(A(.1.),FALSE,C1(.2.),X,Z(.1.));
(*THIS IS THE CARROUT COMPUTATION FOR THE FIRST CELL WITH B,C=0,0*)
                CO(.COUNT.):=C1(.1.);(*FIRST VALUE OF SQ.RT/DIVISION*)
                F(.COUNT.):=COMPF(C1(.1.),P(.COUNT.),X);
  WRITELN('#####':5,'ROW(':5,COUNT:1,')':1,'##########':10);
                TR:=CONV(F(.COUNT.));
                WRITELN('F(':2,COUNT:1,')=':2,TR:1);
                TR:=CONV(CO(.COUNT.));
                WRITELN('CO(':2,COUNT:1,')=':2,TR:1);
(*COMPUTATION OF F,D,E *)
(*NOW GOING TOWARDS THE MIDDLE COMPUTATION CELLS ***************)
(*GENERATING A(J),B(J) AND C(J) FROM A,B,C(J-1) USING DOWNTO *)
                FOR J:=(COLINDX-1) DOWNTO 2 DO
                 BEGIN
                    A(.J.):=COMPS(A(.J.),B(.(J-INDX1).),C1(.(J+1).),
                            F(.COUNT.),X,Z(.J.));
                    TRAN1:=B(.(J-INDX1).);
```

```
                                F(.COUNT.));
                    C(.J.):=COMPE(TRAN1,C(.(J-INDX1).),
                                F(.COUNT.));
(***************SAVEING THE CARRY FOR THE SECOND TIME AREOUND**)
                  IF (COUNT=(MM-1))THEN
                   BEGIN
                    SAV1(.J.):=A(.J.);
                    SAV2(.J.):=B(.J.);
                    SAV3(.J.):=C(.J.)
                   END;
(*NOTE: C1(J) IS CARRY OUT WHILE C1(J+1) IS CARRY IN*************)
(*******P R I N T I N G  MIDDLE CELLS ***************************)
                     WRITELN('          MIDDLE CELL          ');
                TR:=CONV(A(.J.));WRITELN('A(':3,J:1,')':2,TR:1);
                TR:=CONV(B(.J.));WRITELN('B(':3,J:1,')':2,TR:1);
                TR:=CONV(C(.J.));WRITELN('C(':3,J:1,')':2,TR:1);
            TR:=CONV(C1(.(J+1).));WRITELN('C-IN':4,'(':1,J:1,')':2,TR:1);
            TR:=CONV(C1(.J.));WRITELN('C-OUT':4,'(':1,J:1,')':2,TR:1)
                   END;
(*COMPUTATION OF FIRST CELL WITH B,C =0,0 FOR ALL CELL *)
(*EXCEPT FOR THE FIRST ROW WHERE INPUT OF B1,C1 IS MADE*)
IF COLINDX = 3 THEN
   BEGIN(*FOR BOUNDARY ROW CELL IN ROW 1*)
     A(.1.):=COMPS(A(.1.),B(.1.),C1(.2.),F(.COUNT.),X,Z(.1.));(*A<=S*)
     B(.1.):=COMPD(B(.1.),C(.1.),F(.COUNT.));(*B<=D*)
     C(.1.):=COMPE(B(.1.),C(.1.),F(.COUNT.));(*C<=E*)
   END
       ELSE
        BEGIN(*FOR INNER FIRST CELLS,B,C=0,0*)
         A(.1.):=COMPS(A(.1.),FALSE,C1(.2.),F(.COUNT.),X,Z(.1.));
         B(.1.):=COMPD(FALSE,FALSE,F(.COUNT.));(*B<=D*)
         C(.1.):=COMPE(FALSE,FALSE,F(.COUNT.));
         IF (COUNT=(MM-1)) THEN
          BEGIN
           SAV1(.1.):=A(.1.);
           SAV2(.1.):=B(.1.);
           SAV3(.1.):=C(.1.)
          END
        END;
(*P R I N T I N G  F I R S T  C E L L**************************)
               WRITELN('*****':5,' F I R S T  C E L L ':20);
               TR:=CONV(A(.1.));WRITELN('A(1)':4,TR:1);
               TR:=CONV(B(.1.));WRITELN('B(1)':4,TR:1);
               TR:=CONV(C(.1.));WRITELN('C(1)':4,TR:1);
(*NOW GOING FOR THE LAST COLOUMN INDEX*************************)
               A(.COLINDX.):=COMPS(A(.COLINDX.),B(.(COLINDX).),
                                CB,F(.COUNT.),X,Z(.COLINDX.));
               TRAN1:=B(.COLINDX.);
               B(.COLINDX.):=COMPD(TRAN1,C(.(COLINDX)
                                .),F(.COUNT.));
               C(.COLINDX.):=COMPE(TRAN1,
                                C(.(COLINDX).),
                                F(.COUNT.));
(*********************SAVING*****************************)
                IF (COUNT=(MM-1))THEN
                 BEGIN
                  SAV1(.COLINDX.):=A(.COLINDX.);
                  SAV2(.COLINDX.):=B(.COLINDX.);
                  SAV3(.COLINDX.):=C(.COLINDX.)
                 END;
(*******PRINTING THIS LAST CELL IN THE ROW**********************)
              WRITELN('          LAST CELL IN A ROW          ');
              TR:=CONV(A(.COLINDX.));WRITELN('A(',COLINDX,')',TR);
              TR:=CONV(B(.COLINDX.));WRITELN('B(',COLINDX,')',TR);
```

```
313  |  (*****************UPDATE THE INDX POINTERS*****************)
314  |                COLINDX:=COLINDX +2;(*3,5,7,9*)
315  |                INDX1:=1;(*IT WILL REMAIN SO FOR MIDDLE CELLS*)
316  |                COUNT1:=COUNT1 + 1;
317  |                COUNT:= COUNT +1;
318  |                 IF COUNT1= (MM+1) THEN
319  |                  BEGIN(*SECOND TIME AROUND LOOP*)
320  |                   CAROT:=A(.3.);(*NOTE A(.3.) CORROSPONDS TO FIG.*)
321  |                   A(.LL.):=INTOBIN(1);(*ORG VALUE OF A,B,C*)
322  |                   B(.LL.):=INTOBIN(1);
323  |                   C(.LL.):=INTOBIN(1);
324  |  (*************RESTORING THE VALUES OF A,B,C***************)
325  |                   FOR J:= 1 TO (LL-2) DO
326  |                    BEGIN
327  |                     A(.J.):=SAV1(.J.);
328  |                     B(.J.):=SAV2(.J.);
329  |                     C(.J.):=SAV3(.J.)
330  |                    END;
331  |                   COLINDX:=COLINDX-2;
332  |                   CB:=CB OR (CAROT AND Y);(*END AROUND CARRY*)
333  |                   COUNT:=COUNT-1(*SETTING IT BACK TO ORG.COUTNOF5*)
334  |                  END(***SECOND TIME AROUND LOOP*)
335  |            END;(*WHILE COUNT1>6 END*)
336  |  (******************GEWNERATING HARDWARE***********)
337  |  (******************NOTE THAT CARRY OUT DOESNT CHANGE**)
338  |            SUMSIGN:=EXOR(BS,M);
339  |            SUMSIGN:=SUMSIGN AND (NOT CAROT);
340  |            SUMSIGN:=SUMSIGN OR (AS AND CAROT);
341  |            OVF:=CAROT AND (NOT X);
342  |            QAT:=(NOT CAROT) AND X;(*CORRECTOR*)
343  |            INDX1:=(LL +1)DIV 2;
344  |          FOR J:=INDX1 DOWNTO 3 DO(****NOTE THE =>3 TO FIG.*)
345  |           BEGIN
346  |            A(.(2*J-1).):=EXOR( A(.(2*J-1).),QAT);
347  |            TR:=CONV(A(.(2*J-1).));
348  |            WRITELN('SUMBIT':6,'(':1,(2*J-1):1,')':1,TR:1)
349  |           END;
350  |           TR:=CONV(SUMSIGN);WRITELN('SIGN OF SUM':11,TR:2);
351  |           TR:=CONV(OVF);WRITELN('OVERFLOW':9,TR:2);
352  |           TR:=CONV(QAT);WRITELN('CORRECTOR':9,TR:2)
353  |      END (*INITILIZATION*)
354  |  END.(*MAIN PROGRAM TERMINATED*)
355  |  %EOF
Execution begins...
&&&&&&&&&&&&&&&&&&&&&&&&&INPUT SECTION$$$$$$$$$$$$$$$$$
A(1 )0
A(2 )0
A(3 )0
A(4 )0
A(5 )0
A(6 )0
A(7 )1
A(8 )0
A(9 )1
A(10 )0
A(11 )1
B(1 )0
B(2 )0
B(3 )0
B(4 )0
B(5 )0
B(6 )0
B(7 )0
B(8 )0
```

```
B(10 )0
B(11 )1
C(1 )0
C(2 )0
C(3 )0
C(4 )0
C(5 )0
C(6 )0
C(7 )0
C(8 )0
C(9 )0
C(10 )0
C(11 )0
Z(1 )1
Z(2 )1
Z(3 )0
Z(4 )1
Z(5 )0
Z(6 )1
Z(7 )0
Z(8 )1
Z(9 )0
Z(10 )1
Z(11 )0
P(1)0
P(2)0
P(3)0
P(4)0
P(5)0
##### ROW(1)##########
F(1)=1
C01)=0
          MIDDLE CELL
 A(2 )0
 B(2 )0
 C(2 )0
C-IN(2 )0
C-OU(2 )0
***** F I R S T  C E L L
A(1)0
B(1)0
C(1)0
        LAST CELL IN A ROW
A(           3)           0
B(           3)           0
C(           3)           0
##### ROW(2)##########
F(2)=1
C02)=0
          MIDDLE CELL
 A(4 )0
 B(4 )0
 C(4 )0
C-IN(4 )0
C-OU(4 )0
          MIDDLE CELL
 A(3 )0
 B(3 )0
 C(3 )0
C-IN(3 )0
C-OU(3 )0
          MIDDLE CELL
 A(2 )0
 B(2 )0
```

```
C-IN(2 )0
C-OU(2 )0
***** F I R S T  C E L L
A(1)0
B(1)0
C(1)0
          LAST CELL IN A ROW
A(              5)              0
B(              5)              0
C(              5)              0
##### ROW(3)##########
F(3)=1
C03)=0
          MIDDLE CELL
 A(6 )0
 B(6 )0
 C(6 )0
C-IN(6 )0
C-OU(6 )0
          MIDDLE CELL
 A(5 )0
 B(5 )0
 C(5 )0
C-IN(5 )0
C-OU(5 )0
          MIDDLE CELL
 A(4 )0
 B(4 )0
 C(4 )0
C-IN(4 )0
C-OU(4 )0
          MIDDLE CELL
 A(3 )0
 B(3 )0
 C(3 )0
C-IN(3 )0
C-OU(3 )0
          MIDDLE CELL
 A(2 )0
 B(2 )0
 C(2 )0
C-IN(2 )0
C-OU(2 )0
***** F I R S T  C E L L
A(1)0
B(1)0
C(1)0
          LAST CELL IN A ROW
A(              7)              1
B(              7)              0
C(              7)              0
##### ROW(4)##########
F(4)=1
C04)=0
          MIDDLE CELL
 A(8 )0
 B(8 )0
 C(8 )0
C-IN(8 )1
C-OU(8 )1
          MIDDLE CELL
 A(7 )0
 B(7 )0
 C(7 )0
```

```
C-OU(7 )1
            MIDDLE CELL
 A(6 )0
 B(6 )0
 C(6 )0
C-IN(6 )1
C-OU(6 )1
            MIDDLE CELL
 A(5 )1
 B(5 )0
 C(5 )0
C-IN(5 )1
C-OU(5 )0
            MIDDLE CELL
 A(4 )0
 B(4 )0
 C(4 )0
C-IN(4 )0
C-OU(4 )0
            MIDDLE CELL
 A(3 )0
 B(3 )0
 C(3 )0
C-IN(3 )0
C-OU(3 )0
            MIDDLE CELL
 A(2 )0
 B(2 )0
 C(2 )0
C-IN(2 )0
C-OU(2 )0
***** F I R S T  C E L L
A(1)0
B(1)0
C(1)0
          LAST CELL IN A ROW
A(            9)            0
B(            9)            0
C(            9)            0
##### ROW(5)##########
F(5)=1
C05)=0
            MIDDLE CELL
 A(10 )0
 B(10 )0
 C(10 )0
C-IN(10 )1
C-OU(10 )1
            MIDDLE CELL
 A(9 )1
 B(9 )0
 C(9 )0
C-IN(9 )1
C-OU(9 )0
            MIDDLE CELL
 A(8 )0
 B(8 )0
 C(8 )0
C-IN(8 )0
C-OU(8 )0
            MIDDLE CELL
 A(7 )0
 B(7 )0
 C(7 )0
```

```
C-OU(7 )0
                MIDDLE CELL
 A(6 )0
 B(6 )0
 C(6 )0
C-IN(6 )0
C-OU(6 )0
                MIDDLE CELL
 A(5 )1
 B(5 )0
 C(5 )0
C-IN(5 )0
C-OU(5 )0
                MIDDLE CELL
 A(4 )0
 B(4 )0
 C(4 )0
C-IN(4 )0
C-OU(4 )0
                MIDDLE CELL
 A(3 )0
 B(3 )0
 C(3 )0
C-IN(3 )0
C-OU(3 )0
                MIDDLE CELL
 A(2 )0
 B(2 )0
 C(2 )0
C-IN(2 )0
C-OU(2 )0
***** F I R S T  C E L L
A(1)0
B(1)0
C(1)0
            LAST CELL IN A ROW
A(              11)              0
B(              11)              0
C(              11)              0
##### ROW(5)###########
F(5)=1
CO5)=0
                MIDDLE CELL
 A(10 )0
 B(10 )0
 C(10 )0
C-IN(10 )1
C-OU(10 )1
                MIDDLE CELL
 A(9 )1
 B(9 )0
 C(9 )0
C-IN(9 )1
C-OU(9 )0
                MIDDLE CELL
 A(8 )0
 B(8 )0
 C(8 )0
C-IN(8 )0
C-OU(8 )0
                MIDDLE CELL
 A(7 )0
 B(7 )0
 C(7 )0
```

```
C-OU(7 )0
          MIDDLE CELL
 A(6 )0
 B(6 )0
 C(6 )0
C-IN(6 )0
C-OU(6 )0
          MIDDLE CELL
 A(5 )1
 B(5 )0
 C(5 )0
C-IN(5 )0
C-OU(5 )0
          MIDDLE CELL
 A(4 )0
 B(4 )0
 C(4 )0
C-IN(4 )0
C-OU(4 )0
          MIDDLE CELL
 A(3 )0
 B(3 )0
 C(3 )0
C-IN(3 )0
C-OU(3 )0
          MIDDLE CELL
 A(2 )0
 B(2 )0
 C(2 )0
C-IN(2 )0
C-OU(2 )0
***** F I R S T  C E L L
A(1)0
B(1)0
C(1)0
         LAST CELL IN A ROW
A(          11)           0
B(          11)           0
C(          11)           0
SUMBIT(11)0
SUMBIT(9)1
SUMBIT(7)0
SUMBIT(5)1
SIGN OF SUM 0
OVERFLOW    0
CORRECTOR   0

...execution ends
File 'BADDSUB': 355 lines; no diagnostics
20898 bytes of object code generated
2621 statements executed
39032 bytes of memory requested during compilation
5792 bytes returned before execution
10392 bytes requested during execution
```

```
*************************************************************************
        S I M U L A T I O N   P R O G R A M   F O R   T H E
                M U L T I P L Y   O P E R A T I O N
*************************************************************************
WATERLOO PASCAL -- ( CMS, VERSION 3.1 ):  University of Petroleum & Minerals, Dhahran

     1  |  PROGRAM PROCELEM(INPUT,OUTPUT);
     2  |
     3  |
     4  |
     5  |
     6  |
     7  |  CONST
     8  |         X=FALSE;(*X IS THE CONTROL BIT FOR MULT/DIV,SQRT/SQ*)
     9  |         Z=FALSE;(* Z IS THE CONTROL BIT FOR ADD/SUB*)
    10  |                  (*Z=1 ADD/SUB,Z=0 OTHER*)
    11  |         LL=29;(*INCREASING THE ARRAY BIT WISE COLUMN*)
    12  |         MM=14;(*INCREASING THE ARRAY ROW WISE.I.EPBITS*)
    13  |  (*BE REPLACED BY MM LATER*)
    14  |  TYPE
    15  |      SUB1= 1..LL;(*NOTE:THE EXTENSION OF PE WOULD BE DONE BY SUB1,2*)
    16  |      SUB2= 1..MM;
    17  |      MATR1=ARRAY(.SUB1.)OF BOOLEAN;
    18  |      MATR2=ARRAY(.SUB2.)OF BOOLEAN;
    19  |      MTRAN=ARRAY(.SUB1.)OF INTEGER;
    20  |      MTRAN1=ARRAY(.SUB2.)OF INTEGER;
    21  |  VAR
    22  |      A,B,C,C1:MATR1;    (*C1 IS THE CARRY*)
    23  |      F,CO,P:MATR2; (*NOTE:C1(J) IS CARRYOUT WHILE C1(J+1) IS CARRY-IN*)
    24  |                    (*FOR ADDITION/SUBSTRACTION*)
    25  |      COUNT,COLINDX,J,TR,INDX1:INTEGER;
    26  |      A1,B1,C11:MTRAN;
    27  |      P1:MTRAN1;
    28  |      TRAN1:BOOLEAN;
    29  |  (*************************************************************************)
    30  |  (*************************************************************************)
    31  |  (*********************FUNCTION DEFINATIONS*********************************)
    32  |  FUNCTION EXOR(LL1,LL2:BOOLEAN):BOOLEAN;
    33  |    BEGIN
    34  |      EXOR:= (LL1 AND (NOT LL2)) OR ( LL2 AND (NOT LL1))
    35  |    END;
    36  |  (*************************************************************************)
    37  |  FUNCTION COMPS(G1,G2,G3,G4,G5:BOOLEAN):BOOLEAN;
    38  |    VAR
    39  |       OPER:BOOLEAN;
    40  |   BEGIN
    41  |    OPER:=EXOR(G2,G5);
    42  |    OPER:=EXOR(OPER,G1);
    43  |    OPER:=EXOR(OPER,G3);
    44  |    COMPS:=(OPER AND G4) OR (G1 AND (NOT G4))
    45  |   END;(*COMPS FINISHED*)
    46  |  (*************************************************************************)
    47  |  FUNCTION COMPCO (H1,H2,H3,H4:BOOLEAN):BOOLEAN;
    48  |    VAR
    49  |       OPER:BOOLEAN;
    50  |   BEGIN
    51  |    OPER:= EXOR(H2,H4);
    52  |    OPER:= OPER AND (H1 OR H3);
    53  |    COMPCO:= OPER OR ( H1 AND H3);
```

```
 55 |    END;(*COMCO FINISH *)
 56 |   (*************************************************************************)
 57 |   FUNCTION COMPD(P1,P2,P3:BOOLEAN):BOOLEAN;
 58 |    BEGIN
 59 |     COMPD:=(P1 AND P2) OR (P2 AND P3)
 60 |    END;(*COMPD FINISH*)
 61 |   (*************************************************************************)
 62 |   FUNCTION COMPE (Q1,Q2,Q3:BOOLEAN):BOOLEAN;
 63 |    BEGIN
 64 |     COMPE:= Q1 OR ( Q2 AND Q3)
 65 |    END;(*COMPE FINISH *)
 66 |   (*************************************************************************)
 67 |   FUNCTION COMPF (R1,R2,R3:BOOLEAN):BOOLEAN;
 68 |    BEGIN
 69 |     COMPF := (R1 AND R3) OR ( R2 AND (NOT R3))
 70 |    END;(*COMPF FINISHED*)
 71 |   (*************************************************************************)
 72 |   FUNCTION CONV (S1:BOOLEAN):INTEGER;
 73 |    BEGIN
 74 |     IF S1 = TRUE
 75 |             THEN CONV:=1
 76 |             ELSE
 77 |              CONV:=0
 78 |    END;(*CONV FINISHED*)
 79 |   (************FUNCTION TO CONVERT INT TO BINARY********)
 80 |   FUNCTION INTOBIN(VALUE:INTEGER):BOOLEAN;
 81 |    BEGIN
 82 |      IF VALUE = 1
 83 |       THEN INTOBIN:=TRUE
 84 |       ELSE
 85 |       IF VALUE =0
 86 |        THEN INTOBIN:=FALSE
 87 |    END;(*INTOBIN FINISH*)
 88 |   (**********MAIN PROGRAM*********)
 89 |   (*************************************************************************)
 90 |   (*************************************************************************)
 91 |   (*********************M A I N   P R O G R A M*************************)
 92 |   (*READING THE INPUT VALUES WHICH SHOULD BE INPUTED LINE BY LINE********)
 93 |   (*EITHER 1 OR 0,ONE INPUT ONE LINE*************************************)
 94 |   (********NOTE:MSB=A(1) AND LOWER ***************************)
 95 |   (***********************READING LOOP FOR INITIAL VALUE OF A,B,C,P******)
 96 |   BEGIN(****MAIN PROGRAM BEGIN LOOP***************)
 97 |   (*THIS IS THE INPUT SECTION*************************************)
 98 |       A1(.1.):=0;(*THIS BIT NOT TO BE USED IN DIV,ALWYS SET TO 0*)
 99 |       A1(.2.):=0;          (*SEE FIGURE ORIGINAL*)
100 |       A1(.3.):=0;
101 |       A1(.4.):=0;
102 |       A1(.5.):=0;
103 |       A1(.6.):=0;
104 |       A1(.7.):=0;
105 |       A1(.8.):=0;
106 |       A1(.9.):=0;
107 |       A1(.10.):=0;
108 |       A1(.11.):=0;
109 |       A1(.12.):=0;
110 |       A1(.13.):=0;
111 |       A1(.14.):=0;
112 |       A1(.15.):=0;
113 |       A1(.16.):=0;
114 |       A1(.17.):=0;
115 |       A1(.18.):=0;
116 |       A1(.19.):=0;
117 |       A1(.20.):=0;
118 |       A1(.21.):=0;
```

```
120 |       A1(.25.):=0;
121 |       A1(.26.):=0;
122 |       A1(.25.):=0;
123 |       A1(.26.):=0;
124 |       A1(.27.):=0;
125 |       A1(.28.):=0;
126 |       A1(.29.):=0;
127 |   (*****************************************************************)
128 |       B1(.1.):=0;(**)
129 |       B1(.2.):=0;(**)
130 |       B1(.3.):=0;(**)
131 |       B1(.4.):=0;
132 |       B1(.5.):=0;(**)
133 |       B1(.6.):=0;
134 |       B1(.7.):=0;(**)
135 |       B1(.8.):=0;
136 |       B1(.9.):=0;(**)
137 |       B1(.10.):=0;
138 |       B1(.11.):=1;(**)
139 |       B1(.12.):=0;
140 |       B1(.13.):=1;(*LSB*)
141 |       B1(.14.):=0;
142 |       B1(.15.):=0;
143 |       B1(.16.):=0;
144 |       B1(.17.):=0;
145 |       B1(.18.):=0;
146 |       B1(.19.):=0;
147 |       B1(.20.):=0;
148 |       B1(.21.):=0;
149 |       B1(.22.):=0;
150 |       B1(.23.):=0;
151 |       B1(.24.):=0;
152 |       B1(.25.):=0;
153 |       B1(.26.):=0;
154 |       B1(.27.):=0;
155 |       B1(.28.):=0;
156 |       B1(.29.):=0;
157 |   (*****************************************************************)
158 |       C11(.1.):=0;(**)
159 |       C11(.2.):=0;(**)
160 |       C11(.3.):=0;(**)
161 |       C11(.4.):=0;
162 |       C11(.5.):=0;(**)
163 |       C11(.6.):=0;
164 |       C11(.7.):=0;(**)
165 |       C11(.8.):=0;
166 |       C11(.9.):=0;(**)
167 |       C11(.10.):=0;
168 |       C11(.11.):=1;(**)
169 |       C11(.12.):=0;
170 |       C11(.13.):=1;(*LSB*)
171 |       C11(.14.):=0;
172 |       C11(.15.):=0;
173 |       C11(.16.):=0;
174 |       C11(.17.):=0;
175 |       C11(.18.):=0;
176 |       C11(.19.):=0;
177 |       C11(.20.):=0;
178 |       C11(.21.):=0;
179 |       C11(.22.):=0;
180 |       C11(.23.):=0;
181 |       C11(.24.):=0;
182 |       C11(.25.):=0;
183 |       C11(.26.):=0;
```

```
185 |       C11(.28.):=0;
186 |       C11(.29.):=0;
187 |   (*************************************************************)
188 |       P1(.1.):=0;
189 |       P1(.2.):=0;
190 |       P1(.3.):=0;
191 |       P1(.4.):=0;
192 |       P1(.5.):=0;(*ALL ZEROS TILL HERE*)
193 |       P1(.6.):=0;(**PUT ZERO FOR 8 BIT NUMBER*)
194 |       P1(.7.):=0;(*MSB*)
195 |       P1(.8.):=0;(**)
196 |       P1(.9.):=0;(**)
197 |       P1(.10.):=0;(**)
198 |       P1(.11.):=0;(**)
199 |       P1(.12.):=1;(**)
200 |       P1(.13.):=1;(**)
201 |       P1(.14.):=1;(**)
202 |   (*************************************************************************)
203 |     FOR J:=1 TO LL DO
204 |      A(.J.):=INTOBIN(A1(.J.));
205 |   (****NOTICE: THE A,B,C ARE ALL 11 BITS NOW**********************)
206 |     FOR J:=1 TO LL DO
207 |      B(.J.):=INTOBIN(B1(.J.));
208 |   (*********************************************)
209 |     FOR J:=1 TO LL DO
210 |      C(.J.):=INTOBIN(C11(.J.));
211 |   (*********************************************)
212 |     FOR J:=1 TO MM DO
213 |      P(.J.):=INTOBIN(P1(.J.));
214 |   (*********************************************)
215 |   (*B:C:4,6,8,10 ARE DUMMY*)
216 |   (*B:C:5,3,7,9,11 ARE OVERWRITTEN WITH THE LAST CELL COMPUTATION*)
217 |   (*************************************************************************)
218 |   (*************************************************************************)
219 |   (**********************INITIAL READING VALUE FINISHED**************)
220 |      BEGIN(*INITILIZATION*)
221 |         COUNT:=1;
222 |         INDX1:=0;
223 |         COLINDX:=3;
224 |         WHILE COUNT <= MM DO
225 |            BEGIN(*WHILE COUNT*)
226 |               C1(.COLINDX.):= COMPCO(A(.COLINDX.),B(.COLINDX.),X,X);
227 |                  FOR J:=(COLINDX -1) DOWNTO 2 DO
228 |                     BEGIN
229 |                      C1(.J.):=COMPCO(A(.J.),B(.(J-1).),C1(.J+1.),X)
230 |                     END;
231 |                  C1(.1.):=COMPCO(A(.1.),FALSE,C1(.2.),X);
232 |   (*THIS IS THE CARROUT COMPUTATION FOR THE FIRST CELL WITH B,C=0,0*)
233 |                  CO(.COUNT.):=C1(.1.);(*FIRST VALUE OF SQ.RT/DIVISION*)
234 |                  F(.COUNT.):=COMPF(C1(.1.),P(.COUNT.),X);
235 |   (*COMPUTATION OF F,D,E *)
236 |   (*NOW GOING TOWARDS THE MIDDLE COMPUTATION CELLS ***************)
237 |   (*GENERATING A(J),B(J) AND C(J) FROM A,B,C(J-1) USING DOWNTO *)
238 |                  FOR J:=(COLINDX-1) DOWNTO 2 DO
239 |                   BEGIN
240 |                      A(.J.):=COMPS(A(.J.),B(.(J-INDX1).),C1(.(J+1).),
241 |                            F(.COUNT.),X);
242 |                      TRAN1:=B(.(J-INDX1).);
243 |                      B(.J.):=COMPD(TRAN1,C(.(J-INDX1).),
244 |                            F(.COUNT.));
245 |                      C(.J.):=COMPE(TRAN1,C(.(J-INDX1).),
246 |                            F(.COUNT.));
247 |   (*NOTE: C1(J) IS CARRY OUT WHILE C1(J+1) IS CARRY IN*************)
248 |                   END;
```

```
251 |   IF COLINDX = 3 THEN
252 |      BEGIN(*FOR BOUNDARY ROW CELL IN ROW 1*)
253 |        A(.1.):=COMPS(A(.1.),B(.1.),C1(.2.),F(.COUNT.),X);(*A<=S*)
254 |        B(.1.):=COMPD(B(.1.),C(.1.),F(.COUNT.));(*B<=D*)
255 |        C(.1.):=COMPE(B(.1.),C(.1.),F(.COUNT.));(*C<=E*)
256 |      END
257 |          ELSE
258 |           BEGIN(*FOR INNER FIRST CELLS,B,C=0,0*)
259 |                  A(.1.):=COMPS(A(.1.),FALSE,C1(.2.),F(.COUNT.),X);(*A<=S*)
260 |                  B(.1.):=COMPD(FALSE,FALSE,F(.COUNT.));(*B<=D*)
261 |                  C(.1.):=COMPE(FALSE,FALSE,F(.COUNT.))
262 |           END;
263 |  (*P R I N T I N G  F I R S T  C E L L**************************)
264 |  (*NOW GOING FOR THE LAST COLOUMN INDEX*****************************)
265 |                  A(.COLINDX.):=COMPS(A(.COLINDX.),B(.(COLINDX).),
266 |                                      X,F(.COUNT.),X);
267 |                  TRAN1:=B(.COLINDX.);
268 |                  B(.COLINDX.):=COMPD(TRAN1,C(.(COLINDX)
269 |                                      .),F(.COUNT.));
270 |                  C(.COLINDX.):=COMPE(TRAN1,
271 |                                      C(.(COLINDX).),
272 |                                      F(.COUNT.));
273 |  (*******PRINTING THIS LAST CELL IN THE ROW************************)
274 |  (**********************UPDATE THE INDX POINTERS**************)
275 |              COLINDX:=COLINDX +2;(*3,5,7,9*)
276 |              INDX1:=1;(*IT WILL REMAIN SO FOR MIDDLE CELLS*)
277 |              COUNT:=COUNT+1
278 |           END;(*WHILE COUNT >5 END*)
279 |       IF (X=FALSE)THEN
280 |        BEGIN
281 |         FOR J:=1 TO LL DO
282 |          BEGIN
283 |           TR:=CONV(A(.J.));
284 |           WRITELN('OUTPRD':6,'(':1,J:1,')':1,TR:1)
285 |          END
286 |        END
287 |       ELSE
288 |        BEGIN
289 |         FOR J:=1 TO MM DO
290 |          BEGIN
291 |           TR:=CONV(CO(.J.));
292 |           WRITELN('OUTDIV':6,'(':1,J:1,')':1,TR:1)
293 |          END
294 |        END;
295 |     END (*INITILIZATION*)
296 |  END.(*MAIN PROGRAM TERMINATED*)
297 |  (*READ 11A'S,B'S AND C'S AND 5 P'S*)
298 |  %EOF
Execution begins...
OUTPRD(1)0
OUTPRD(2)0
OUTPRD(3)0
OUTPRD(4)0
OUTPRD(5)0
OUTPRD(6)0
OUTPRD(7)0
OUTPRD(8)0
OUTPRD(9)0
OUTPRD(10)0
OUTPRD(11)0
OUTPRD(12)0
OUTPRD(13)0
OUTPRD(14)0
```

```
OUTPRD(16)0
OUTPRD(17)1
OUTPRD(18)0
OUTPRD(19)1
OUTPRD(20)0
OUTPRD(21)1
OUTPRD(22)0
OUTPRD(23)0
OUTPRD(24)0
OUTPRD(25)0
OUTPRD(26)0
OUTPRD(27)0
OUTPRD(28)0
OUTPRD(29)0

...execution ends
File 'EXTCPROC': 298 lines; no diagnostics
15121 bytes of object code generated
5561 statements executed
30016 bytes of memory requested during compilation
5792 bytes returned before execution
10392 bytes requested during execution
```

```
**********************************************************************
        S I M U L A T I O N  P R O G R A M  F O R  T H E

              D I V I S I O N  O P E R A T I O N
**********************************************************************

WATERLOO PASCAL -- ( CMS, VERSION 3.1 ):  University of Petroleum & Minerals, Dhahran

     1 |  PROGRAM PROCELEM(INPUT,OUTPUT);
     2 |
     3 |
     4 |
     5 |
     6 |
     7 |  CONST
     8 |         X=TRUE;(*X IS THE CONTROL BIT FOR MULT/DIV,SQRT/SQ*)
     9 |         Z=FALSE;(* Z IS THE CONTROL BIT FOR ADD/SUB*)
    10 |                  (*Z=1 ADD/SUB,Z=0 OTHER*)
    11 |         LL=29;(*INCREASING THE ARRAY BIT WISE COLUMN*)
    12 |         MM=14;(*INCREASING THE ARRAY ROW WISE.I.EPBITS*)
    13 |  (*BE REPLACED BY MM LATER*)
    14 |  TYPE
    15 |      SUB1= 1..LL;(*NOTE:THE EXTENSION OF PE WOULD BE DONE BY SUB1,2*)
    16 |      SUB2= 1..MM;
    17 |      MATR1=ARRAY(.SUB1.)OF BOOLEAN;
    18 |      MATR2=ARRAY(.SUB2.)OF BOOLEAN;
    19 |      MTRAN=ARRAY(.SUB1.)OF INTEGER;
    20 |      MTRAN1=ARRAY(.SUB2.)OF INTEGER;
    21 |  VAR
    22 |      A,B,C,C1:MATR1;    (*C1 IS THE CARRY*)
    23 |      F,CO,P:MATR2; (*NOTE:C1(J) IS CARRYOUT WHILE C1(J+1) IS CARRY-IN*)
    24 |                     (*FOR ADDITION/SUBSTRACTION*)
    25 |      COUNT,COLINDX,J,TR,INDX1:INTEGER;
    26 |      A1,B1,C11:MTRAN;
    27 |      P1:MTRAN1;
    28 |      TRAN1:BOOLEAN;
    29 |  (**********************************************************************)
    30 |  (**********************************************************************)
    31 |  (********************FUNCTION DEFINATIONS**************************)
    32 |  FUNCTION EXOR(LL1,LL2:BOOLEAN):BOOLEAN;
    33 |    BEGIN
    34 |      EXOR:= (LL1 AND (NOT LL2)) OR ( LL2 AND (NOT LL1))
    35 |    END;
    36 |  (**********************************************************************)
    37 |  FUNCTION COMPS(G1,G2,G3,G4,G5:BOOLEAN):BOOLEAN;
    38 |    VAR
    39 |       OPER:BOOLEAN;
    40 |   BEGIN
    41 |    OPER:=EXOR(G2,G5);
    42 |    OPER:=EXOR(OPER,G1);
    43 |    OPER:=EXOR(OPER,G3);
    44 |    COMPS:=(OPER AND G4) OR (G1 AND (NOT G4))
    45 |   END;(*COMPS FINISHED*)
    46 |  (**********************************************************************)
    47 |  FUNCTION COMPCO (H1,H2,H3,H4:BOOLEAN):BOOLEAN;
    48 |    VAR
    49 |       OPER:BOOLEAN;
    50 |   BEGIN
    51 |    OPER:= EXOR(H2,H4);
    52 |    OPER:= OPER AND (H1 OR H3);
```

```
54 |    OPER:= OPER OR ( H1 AND H3)
55 |   END;(*COMCO FINISH *)
56 |  (*************************************************************************)
57 |  FUNCTION COMPD(P1,P2,P3:BOOLEAN):BOOLEAN;
58 |   BEGIN
59 |    COMPD:=(P1 AND P2) OR (P2 AND P3)
60 |   END;(*COMPD FINISH*)
61 |  (*************************************************************************)
62 |  FUNCTION COMPE (Q1,Q2,Q3:BOOLEAN):BOOLEAN;
63 |   BEGIN
64 |    COMPE:= Q1 OR ( Q2 AND Q3)
65 |   END;(*COMPE FINISH *)
66 |  (*************************************************************************)
67 |  FUNCTION COMPF (R1,R2,R3:BOOLEAN):BOOLEAN;
68 |   BEGIN
69 |    COMPF := (R1 AND R3) OR ( R2 AND (NOT R3))
70 |   END;(*COMPF FINISHED*)
71 |  (*************************************************************************)
72 |  FUNCTION CONV (S1:BOOLEAN):INTEGER;
73 |   BEGIN
74 |    IF S1 = TRUE
75 |            THEN CONV:=1
76 |            ELSE
77 |             CONV:=0
78 |   END;(*CONV FINISHED*)
79 |  (*************FUNCTION TO CONVERT INT TO BINARY********)
80 |  FUNCTION INTOBIN(VALUE:INTEGER):BOOLEAN;
81 |   BEGIN
82 |    IF VALUE = 1
83 |       THEN INTOBIN:=TRUE
84 |       ELSE
85 |       IF VALUE =0
86 |        THEN INTOBIN:=FALSE
87 |   END;(*INTOBIN FINISH*)
88 |  (**********MAIN PROGRAM**********)
89 |  (*************************************************************************)
90 |  (*************************************************************************)
91 |  (*********************M A I N   P R O G R A M*****************************)
92 |  (*READING THE INPUT VALUES WHICH SHOULD BE INPUTED LINE BY LINE********)
93 |  (*EITHER 1 OR 0,ONE INPUT ONE LINE*************************************)
94 |  (********NOTE:MSB=A(1) AND LOWER ***************************)
95 |  (************************READING LOOP FOR INITIAL VALUE OF A,B,C,P******)
96 |  BEGIN(****MAIN PROGRAM BEGIN LOOP***************)
97 |  (*THIS IS THE INPUT SECTION*********************************)
98 |      A1(.1.):=0;(*THIS BIT NOT TO BE USED IN DIV,ALWYS SET TO 0*)
99 |      A1(.2.):=0;          (*SEE FIGURE ORIGINAL*)
100|      A1(.3.):=0;
101|      A1(.4.):=0;
102|      A1(.5.):=0;
103|      A1(.6.):=0;
104|      A1(.7.):=0;
105|      A1(.8.):=0;
106|      A1(.9.):=0;
107|      A1(.10.):=0;
108|      A1(.11.):=0;
109|      A1(.12.):=0;
110|      A1(.13.):=1;
111|      A1(.14.):=1;
112|      A1(.15.):=1;
113|      A1(.16.):=1;
114|      A1(.17.):=1;
115|      A1(.18.):=1;
116|      A1(.19.):=1;
117|      A1(.20.):=1;
```

```
120  |       A1(.23.):=0;
121  |       A1(.24.):=0;
122  |       A1(.25.):=0;
123  |       A1(.26.):=0;
124  |       A1(.27.):=0;
125  |       A1(.28.):=0;
126  |       A1(.29.):=0;
127  |   (*************************************************************)
128  |       B1(.1.):=0;(**)
129  |       B1(.2.):=0;(**)
130  |       B1(.3.):=0;(**)
131  |       B1(.4.):=0;
132  |       B1(.5.):=0;(**)
133  |       B1(.6.):=0;
134  |       B1(.7.):=0;(**)
135  |       B1(.8.):=0;
136  |       B1(.9.):=0;(**)
137  |       B1(.10.):=0;
138  |       B1(.11.):=1;(**)
139  |       B1(.12.):=0;
140  |       B1(.13.):=0;(*LSB*)
141  |       B1(.14.):=0;
142  |       B1(.15.):=0;
143  |       B1(.16.):=0;
144  |       B1(.17.):=0;
145  |       B1(.18.):=0;
146  |       B1(.19.):=0;
147  |       B1(.20.):=0;
148  |       B1(.21.):=0;
149  |       B1(.22.):=0;
150  |       B1(.23.):=0;
151  |       B1(.24.):=0;
152  |       B1(.25.):=0;
153  |       B1(.26.):=0;
154  |       B1(.27.):=0;
155  |       B1(.28.):=0;
156  |       B1(.29.):=0;
157  |   (*************************************************************)
158  |       C11(.1.):=0;(**)
159  |       C11(.2.):=0;(**)
160  |       C11(.3.):=0;(**)
161  |       C11(.4.):=0;
162  |       C11(.5.):=0;(**)
163  |       C11(.6.):=0;
164  |       C11(.7.):=0;(**)
165  |       C11(.8.):=0;
166  |       C11(.9.):=0;(**)
167  |       C11(.10.):=0;
168  |       C11(.11.):=1;(**)
169  |       C11(.12.):=0;
170  |       C11(.13.):=0;(*LSB*)
171  |       C11(.14.):=0;
172  |       C11(.15.):=0;
173  |       C11(.16.):=0;
174  |       C11(.17.):=0;
175  |       C11(.18.):=0;
176  |       C11(.19.):=0;
177  |       C11(.20.):=0;
178  |       C11(.21.):=0;
179  |       C11(.22.):=0;
180  |       C11(.23.):=0;
181  |       C11(.24.):=0;
182  |       C11(.25.):=0;
```

```
184 |       C11(.27.):=0;
185 |       C11(.28.):=0;
186 |       C11(.29.):=0;
187 |   (*********************************************************************)
188 |       P1(.1.):=0;
189 |       P1(.2.):=0;
190 |       P1(.3.):=0;
191 |       P1(.4.):=0;
192 |       P1(.5.):=0;(*ALL ZEROS TILL HERE*)
193 |       P1(.6.):=0;
194 |       P1(.7.):=0;
195 |       P1(.8.):=0;
196 |       P1(.9.):=0;
197 |       P1(.10.):=0;
198 |       P1(.11.):=0;
199 |       P1(.12.):=0;
200 |       P1(.13.):=0;
201 |       P1(.14.):=0;
202 |   (*************************************************************************)
203 |     FOR J:=1 TO LL DO
204 |      A(.J.):=INTOBIN(A1(.J.));
205 |   (****NOTICE: THE A,B,C ARE ALL 11 BITS NOW************************)
206 |     FOR J:=1 TO LL DO
207 |      B(.J.):=INTOBIN(B1(.J.));
208 |   (*********************************************)
209 |     FOR J:=1 TO LL DO
210 |      C(.J.):=INTOBIN(C11(.J.));
211 |   (*********************************************)
212 |     FOR J:=1 TO MM DO
213 |      P(.J.):=INTOBIN(P1(.J.));
214 |   (*********************************************)
215 |   (*B:C:4,6,8,10 ARE DUMMY*)
216 |   (*B:C:5,3,7,9,11 ARE OVERWRITTEN WITH THE LAST CELL COMPUTATION*)
217 |   (*************************************************************************)
218 |   (*************************************************************************)
219 |   (************************INITIAL READING VALUE FINISHED***************)
220 |      BEGIN(*INITILIZATION*)
221 |         COUNT:=1;
222 |         INDX1:=0;
223 |         COLINDX:=3;
224 |         WHILE COUNT <= MM DO
225 |            BEGIN(*WHILE COUNT*)
226 |               C1(.COLINDX.):= COMPCO(A(.COLINDX.),B(.COLINDX.),X,X);
227 |                  FOR J:=(COLINDX -1) DOWNTO 2 DO
228 |                     BEGIN
229 |                      C1(.J.):=COMPCO(A(.J.),B(.(J-1).),C1(.J+1.),X)
230 |                     END;
231 |                  C1(.1.):=COMPCO(A(.1.),FALSE,C1(.2.),X);
232 |   (*THIS IS THE CARROUT COMPUTATION FOR THE FIRST CELL WITH B,C=0,0*)
233 |                  CO(.COUNT.):=C1(.1.);(*FIRST VALUE OF SQ.RT/DIVISION*)
234 |                  F(.COUNT.):=COMPF(C1(.1.),P(.COUNT.),X);
235 |   (*COMPUTATION OF F,D,E *)
236 |   (*NOW GOING TOWARDS THE MIDDLE COMPUTATION CELLS ***************)
237 |   (*GENERATING A(J),B(J) AND C(J) FROM A,B,C(J-1) USING DOWNTO *)
238 |                  FOR J:=(COLINDX-1) DOWNTO 2 DO
239 |                   BEGIN
240 |                      A(.J.):=COMPS(A(.J.),B(.(J-INDX1).),C1(.(J+1).),
241 |                            F(.COUNT.),X);
242 |                      TRAN1:=B(.(J-INDX1).);
243 |                      B(.J.):=COMPD(TRAN1,C(.(J-INDX1).),
244 |                            F(.COUNT.));
245 |                      C(.J.):=COMPE(TRAN1,C(.(J-INDX1).),
246 |                            F(.COUNT.));
247 |   (*NOTE: C1(J) IS CARRY OUT WHILE C1(J+1) IS CARRY IN**************)
```

```
249  |  (*COMPUTATION OF FIRST CELL WITH B,C =0,0 FOR ALL CELL *)
250  |  (*EXCEPT FOR THE FIRST ROW WHERE INPUT OF B1,C1 IS MADE*)
251  |  IF COLINDX = 3 THEN
252  |     BEGIN(*FOR BOUNDARY ROW CELL IN ROW 1*)
253  |       A(.1.):=COMPS(A(.1.),B(.1.),C1(.2.),F(.COUNT.),X);(*A<=S*)
254  |       B(.1.):=COMPD(B(.1.),C(.1.),F(.COUNT.));(*B<=D*)
255  |       C(.1.):=COMPE(B(.1.),C(.1.),F(.COUNT.));(*C<=E*)
256  |     END
257  |         ELSE
258  |          BEGIN(*FOR INNER FIRST CELLS,B,C=0,0*)
259  |                A(.1.):=COMPS(A(.1.),FALSE,C1(.2.),F(.COUNT.),X);(*A<=S*)
260  |                B(.1.):=COMPD(FALSE,FALSE,F(.COUNT.));(*B<=D*)
261  |                C(.1.):=COMPE(FALSE,FALSE,F(.COUNT.))
262  |          END;
263  |  (*P R I N T I N G  F I R S T  C E L L**************************)
264  |  (*NOW GOING FOR THE LAST COLOUMN INDEX**************************)
265  |                A(.COLINDX.):=COMPS(A(.COLINDX.),B(.(COLINDX).),
266  |                                X,F(.COUNT.),X);
267  |                TRAN1:=B(.COLINDX.);
268  |                B(.COLINDX.):=COMPD(TRAN1,C(.(COLINDX)
269  |                                .),F(.COUNT.));
270  |                C(.COLINDX.):=COMPE(TRAN1,
271  |                                C(.(COLINDX).),
272  |                                F(.COUNT.));
273  |  (*******PRINTING THIS LAST CELL IN THE ROW**********************)
274  |  (**********************UPDATE THE INDX POINTERS**************)
275  |              COLINDX:=COLINDX +2;(*3,5,7,9*)
276  |              INDX1:=1;(*IT WILL REMAIN SO FOR MIDDLE CELLS*)
277  |              COUNT:=COUNT+1
278  |           END;(*WHILE COUNT >5 END*)
279  |       IF (X=FALSE)THEN
280  |        BEGIN
281  |         FOR J:=1 TO LL DO
282  |          BEGIN
283  |           TR:=CONV(A(.J.));
284  |           WRITELN('OUTPRD':6,'(':1,J:1,')':1,TR:1)
285  |          END
286  |        END
287  |       ELSE
288  |        BEGIN
289  |         FOR J:=1 TO MM DO
290  |          BEGIN
291  |           TR:=CONV(CO(.J.));
292  |           WRITELN('OUTDIV':6,'(':1,J:1,')':1,TR:1)
293  |          END
294  |        END;
295  |     END (*INITILIZATION*)
296  |  END.(*MAIN PROGRAM TERMINATED*)
297  |  (*READ 11A'S,B'S AND C'S AND 5 P'S*)
298  |  %EOF
Execution begins...
OUTDIV(1)1
OUTDIV(2)1
OUTDIV(3)1
OUTDIV(4)1
OUTDIV(5)0
OUTDIV(6)0
OUTDIV(7)1
OUTDIV(8)1
OUTDIV(9)1
OUTDIV(10)1
OUTDIV(11)1
OUTDIV(12)1
OUTDIV(13)1
```

```
...execution ends
File 'EXDIV': 298 lines; no diagnostics
15121 bytes of object code generated
5482 statements executed
30016 bytes of memory requested during compilation
5792 bytes returned before execution
10392 bytes requested during execution
```

```
**********************************************************************

      S I M U L A T I O N   P R O G R A M   F O R   T H E
              S Q U A R E   O P E R A T I O N

**********************************************************************
WATERLOO PASCAL -- ( CMS, VERSION 3.1 ):  University of Petroleum & Minerals, Dhahran

  1 |  PROGRAM PROCELEM( INPUT,OUTPUT);
  2 |  (* SIMULATION PROGRAM FOR THE SQUARE AND SQUARE-ROOT OPERATION******)
  3 |  (* FOR SQUARE OPERATION:ALL THE A'S ARE ZERO AND X SHOULD BE FALSE*)
  4 |  (* THE OUTPUT IS GOTTON FROM BOTTOM LSB MOST RIGHT.*****************)
  5 |  (**FOR THE SQUARE ROOT OPERATION: ALL THE P'S ARE ZERO THE NUMBER**)
  6 |  (*** FEED BY THE A'S:A CAN BE 28-BIT NUMBER,OUTPUT 14 BIT P'S**)
  7 |  (****THE X HAS TO BE TRUE************************)
  8 |  CONST
  9 |        X=FALSE;(*X IS THE CONTROL BIT FOR MULT/DIV,SQRT/SQ*)
 10 |        LL=29;(*INCREASING THE ARRAY BIT WISE COLUMN*)
 11 |        MM=14;(*INCREASING THE ARRAY ROW WISE.I.EPBITS*)
 12 |  (*BE REPLACED BY MM LATER*)
 13 |  TYPE
 14 |      SUB1= 1..LL;(*NOTE:THE EXTENSION OF PE WOULD BE DONE BY SUB1,2*)
 15 |      SUB2= 1..MM;
 16 |      MATR1=ARRAY(.SUB1.)OF BOOLEAN;
 17 |      MATR2=ARRAY(.SUB2.)OF BOOLEAN;
 18 |      MTRAN=ARRAY(.SUB1.)OF INTEGER;
 19 |      MTRAN1=ARRAY(.SUB2.)OF INTEGER;
 20 |  VAR
 21 |      A,B,C,C1:MATR1;    (*C1 IS THE CARRY*)
 22 |      F,CO,P:MATR2; (*NOTE:C1(J) IS CARRYOUT WHILE C1(J+1) IS CARRY-IN*)
 23 |                     (*FOR ADDITION/SUBSTRACTION*)
 24 |      COUNT,COLINDX,J,TR,INDX1:INTEGER;
 25 |      A1,B1,C11:MTRAN;
 26 |      P1:MTRAN1;
 27 |      TRAN1:BOOLEAN;
 28 |  (**********************************************************************)
 29 |  (**********************************************************************)
 30 |  (********************FUNCTION DEFINATIONS*****************************)
 31 |  FUNCTION EXOR(LL1,LL2:BOOLEAN):BOOLEAN;
 32 |    BEGIN
 33 |      EXOR:= (LL1 AND (NOT LL2)) OR ( LL2 AND (NOT LL1))
 34 |    END;
 35 |  (**********************************************************************)
 36 |  FUNCTION COMPS(G1,G2,G3,G4,G5:BOOLEAN):BOOLEAN;
 37 |    VAR
 38 |       OPER:BOOLEAN;
 39 |   BEGIN
 40 |    OPER:=EXOR(G2,G5);
 41 |    OPER:=EXOR(OPER,G1);
 42 |    OPER:=EXOR(OPER,G3);
 43 |    COMPS:=(OPER AND G4) OR (G1 AND (NOT G4))
 44 |   END;(*COMPS FINISHED*)
 45 |  (**********************************************************************)
 46 |  FUNCTION COMPCO (H1,H2,H3,H4:BOOLEAN):BOOLEAN;
 47 |    VAR
 48 |       OPER:BOOLEAN;
 49 |   BEGIN
 50 |    OPER:= EXOR(H2,H4);
 51 |    OPER:= OPER AND (H1 OR H3);
 52 |    COMPCO:= OPER OR ( H1 AND H3);
 53 |    OPER:= OPER OR ( H1 AND H3)
 54 |   END;(*COMCO FINISH *)
```

```
  FUNCTION COMPD(P1,P2,P3:BOOLEAN):BOOLEAN;
   BEGIN
    COMPD:=(P1 AND P2) OR (P2 AND P3)
   END;(*COMPD FINISH*)
  (*************************************************************************)
  FUNCTION COMPE (Q1,Q2,Q3:BOOLEAN):BOOLEAN;
   BEGIN
    COMPE:= Q1 OR ( Q2 AND Q3)
   END;(*COMPE FINISH *)
  (*************************************************************************)
  FUNCTION COMPF (R1,R2,R3:BOOLEAN):BOOLEAN;
   BEGIN
    COMPF := (R1 AND R3) OR ( R2 AND (NOT R3))
   END;(*COMPF FINISHED*)
  (*************************************************************************)
  FUNCTION CONV (S1:BOOLEAN):INTEGER;
   BEGIN
    IF S1 = TRUE
            THEN CONV:=1
            ELSE
             CONV:=0
   END;(*CONV FINISHED*)
  (************FUNCTION TO CONVERT INT TO BINARY********)
  FUNCTION INTOBIN(VALUE:INTEGER):BOOLEAN;
   BEGIN
     IF VALUE = 1
       THEN INTOBIN:=TRUE
       ELSE
       IF VALUE =0
        THEN INTOBIN:=FALSE
   END;(*INTOBIN FINISH*)
  (*********MAIN PROGRAM*********)
  (*************************************************************************)
  (*************************************************************************)
  (**********************M A I N   P R O G R A M**************************)
  (*READING THE INPUT VALUES WHICH SHOULD BE INPUTED LINE BY LINE********)
  (*EITHER 1 OR 0,ONE INPUT ONE LINE*************************************)
  (********NOTE:MSB=A(1) AND LOWER ***************************)
  (************************READING LOOP FOR INITIAL VALUE OF A,B,C,P*******)
  BEGIN(****MAIN PROGRAM BEGIN LOOP***************)
  (*THIS IS THE INPUT SECTION*************************************)
      A1(.1.):=0;(*THIS BIT NOT TO BE USED IN DIV,ALWYS SET TO 0*)
      A1(.2.):=0;          (*SEE FIGURE ORIGINAL*)
      A1(.3.):=0;
      A1(.4.):=0;
      A1(.5.):=0;
      A1(.6.):=0;
      A1(.7.):=0;
      A1(.8.):=0;
      A1(.9.):=0;
      A1(.10.):=0;
      A1(.11.):=0;
      A1(.12.):=0;
      A1(.13.):=0;
      A1(.14.):=0;
      A1(.15.):=0;
      A1(.16.):=0;
      A1(.17.):=0;
      A1(.18.):=0;
      A1(.19.):=0;
      A1(.20.):=0;
      A1(.21.):=0;
      A1(.22.):=0;
      A1(.23.):=0;
```

```
122 |       A1(.26.):=0;
123 |       A1(.27.):=0;
124 |       A1(.28.):=0;
125 |       A1(.29.):=0;(**LSB**)
126 |   (*************************************************************)
127 |       B1(.1.):=1;(**)
128 |       B1(.2.):=0;(**)
129 |       B1(.3.):=1;(**)
130 |       B1(.4.):=0;
131 |       B1(.5.):=1;(**)
132 |       B1(.6.):=0;
133 |       B1(.7.):=1;(**)
134 |       B1(.8.):=0;
135 |       B1(.9.):=1;(**)
136 |       B1(.10.):=0;
137 |       B1(.11.):=1;(**)
138 |       B1(.12.):=0;
139 |       B1(.13.):=1;(**)
140 |       B1(.14.):=0;
141 |       B1(.15.):=1;(**)
142 |       B1(.16.):=0;
143 |       B1(.17.):=1;(**)
144 |       B1(.18.):=0;
145 |       B1(.19.):=1;(**)
146 |       B1(.20.):=0;
147 |       B1(.21.):=1;(**)
148 |       B1(.22.):=0;
149 |       B1(.23.):=1;(**)
150 |       B1(.24.):=0;
151 |       B1(.25.):=1;(**)
152 |       B1(.26.):=0;
153 |       B1(.27.):=1;(**)
154 |       B1(.28.):=0;
155 |       B1(.29.):=1;(**)
156 |   (*************************************************************)
157 |       C11(.1.):=0;(**)
158 |       C11(.2.):=1;(**)
159 |       C11(.3.):=0;(**)
160 |       C11(.4.):=0;
161 |       C11(.5.):=0;(**)
162 |       C11(.6.):=0;
163 |       C11(.7.):=0;(**)
164 |       C11(.8.):=0;
165 |       C11(.9.):=0;(**)
166 |       C11(.10.):=0;
167 |       C11(.11.):=0;(**)
168 |       C11(.12.):=0;
169 |       C11(.13.):=0;(**)
170 |       C11(.14.):=0;
171 |       C11(.15.):=0;(**)
172 |       C11(.16.):=0;
173 |       C11(.17.):=0;(**)
174 |       C11(.18.):=0;
175 |       C11(.19.):=0;(**)
176 |       C11(.20.):=0;
177 |       C11(.21.):=0;(**)
178 |       C11(.22.):=0;
179 |       C11(.23.):=0;(**)
180 |       C11(.24.):=0;
181 |       C11(.25.):=0;(**)
182 |       C11(.26.):=0;
183 |       C11(.27.):=0;(**)
184 |       C11(.28.):=0;
```

```
187 |       P1(.1.):=0;
188 |       P1(.2.):=0;
189 |       P1(.3.):=0;
190 |       P1(.4.):=0;
191 |       P1(.5.):=0;
192 |       P1(.6.):=0;
193 |       P1(.7.):=0;
194 |       P1(.8.):=0;
195 |       P1(.9.):=0;
196 |       P1(.10.):=0;
197 |       P1(.11.):=1;
198 |       P1(.12.):=0;
199 |       P1(.13.):=0;
200 |       P1(.14.):=1;(*****LEAST SIGNIFICANT BIT***)
201 |   (*************************************************************************)
202 |     FOR J:=1 TO LL DO
203 |      A(.J.):=INTOBIN(A1(.J.));
204 |   (****NOTICE: THE A,B,C ARE ALL 11 BITS NOW************************)
205 |     FOR J:=1 TO LL DO
206 |      B(.J.):=INTOBIN(B1(.J.));
207 |   (*****************************************)
208 |     FOR J:=1 TO LL DO
209 |      C(.J.):=INTOBIN(C11(.J.));
210 |   (*****************************************)
211 |     FOR J:=1 TO MM DO
212 |      P(.J.):=INTOBIN(P1(.J.));
213 |   (*****************************************)
214 |   (*B:C:4,6,8,10 ARE DUMMY*)
215 |   (*B:C:5,3,7,9,11 ARE OVERWRITTEN WITH THE LAST CELL COMPUTATION*)
216 |   (*************************************************************************)
217 |   (*************************************************************************)
218 |   (************************INITIAL READING VALUE FINISHED***************)
219 |      BEGIN(*INITILIZATION*)
220 |         COUNT:=1;
221 |         INDX1:=0;
222 |         COLINDX:=3;
223 |         WHILE COUNT <= MM DO
224 |            BEGIN(*WHILE COUNT*)
225 |               C1(.COLINDX.):= COMPCO(A(.COLINDX.),B(.COLINDX.),X,X);
226 |                  FOR J:=(COLINDX -1) DOWNTO 2 DO
227 |                     BEGIN
228 |                      C1(.J.):=COMPCO(A(.J.),B(.(J-1).),C1(.J+1.),X)
229 |                     END;
230 |                  C1(.1.):=COMPCO(A(.1.),FALSE,C1(.2.),X);
231 |   (*THIS IS THE CARROUT COMPUTATION FOR THE FIRST CELL WITH B,C=0,0*)
232 |                  CO(.COUNT.):=C1(.1.);(*FIRST VALUE OF SQ.RT/DIVISION*)
233 |                  F(.COUNT.):=COMPF(C1(.1.),P(.COUNT.),X);
234 |   (*COMPUTATION OF F,D,E *)
235 |   (*NOW GOING TOWARDS THE MIDDLE COMPUTATION CELLS ***************)
236 |   (*GENERATING A(J),B(J) AND C(J) FROM A,B,C(J-1) USING DOWNTO *)
237 |                  FOR J:=(COLINDX-1) DOWNTO 2 DO
238 |                   BEGIN
239 |                      A(.J.):=COMPS(A(.J.),B(.(J-INDX1).),C1(.(J+1).),
240 |                              F(.COUNT.),X);
241 |                      TRAN1:=B(.(J-INDX1).);
242 |                      B(.J.):=COMPD(TRAN1,C(.(J-INDX1).),
243 |                              F(.COUNT.));
244 |                      C(.J.):=COMPE(TRAN1,C(.(J-INDX1).),
245 |                              F(.COUNT.));
246 |   (*NOTE: C1(J) IS CARRY OUT WHILE C1(J+1) IS CARRY IN*************)
247 |                   END;
248 |   (*COMPUTATION OF FIRST CELL WITH B,C =0,0 FOR ALL CELL *)
249 |   (*EXCEPT FOR THE FIRST ROW WHERE INPUT OF B1,C1 IS MADE*)
```

```
251 |       BEGIN(*FOR BOUNDARY ROW CELL IN ROW 1*)
252 |         A(.1.):=COMPS(A(.1.),B(.1.),C1(.2.),F(.COUNT.),X);(*A<=S*)
253 |         B(.1.):=COMPD(B(.1.),C(.1.),F(.COUNT.));(*B<=D*)
254 |         C(.1.):=COMPE(B(.1.),C(.1.),F(.COUNT.));(*C<=E*)
255 |       END
256 |           ELSE
257 |            BEGIN(*FOR INNER FIRST CELLS,B,C=0,0*)
258 |                   A(.1.):=COMPS(A(.1.),FALSE,C1(.2.),F(.COUNT.),X);(*A<=S*)
259 |                   B(.1.):=COMPD(FALSE,FALSE,F(.COUNT.));(*B<=D*)
260 |                   C(.1.):=COMPE(FALSE,FALSE,F(.COUNT.))
261 |            END;
262 |   (*P R I N T I N G  F I R S T  C E L L**************************)
263 |   (*NOW GOING FOR THE LAST COLOUMN INDEX*************************)
264 |                   A(.COLINDX.):=COMPS(A(.COLINDX.),B(.(COLINDX).),
265 |                                       X,F(.COUNT.),X);
266 |                   TRAN1:=B(.COLINDX.);
267 |                   B(.COLINDX.):=COMPD(TRAN1,C(.(COLINDX)
268 |                                       .),F(.COUNT.));
269 |                   C(.COLINDX.):=COMPE(TRAN1,
270 |                                       C(.(COLINDX).),
271 |                                       F(.COUNT.));
272 |   (*******PRINTING THIS LAST CELL IN THE ROW**********************)
273 |   (************************UPDATE THE INDX POINTERS**************)
274 |                COLINDX:=COLINDX +2;(*3,5,7,9*)
275 |                INDX1:=1;(*IT WILL REMAIN SO FOR MIDDLE CELLS*)
276 |                COUNT:=COUNT+1
277 |            END;(*WHILE COUNT >5 END*)
278 |       IF (X=FALSE)THEN
279 |        BEGIN
280 |         FOR J:=1 TO LL DO
281 |          BEGIN
282 |           TR:=CONV(A(.J.));
283 |           WRITELN('OUTSQR':6,'(':1,J:1,')':1,TR:1)
284 |          END
285 |        END
286 |       ELSE
287 |        BEGIN
288 |         FOR J:=1 TO MM DO
289 |          BEGIN
290 |           TR:=CONV(CO(.J.));
291 |           WRITELN('OUTSRT':6,'(':1,J:1,')':1,TR:1)
292 |          END
293 |        END;
294 |     END (*INITILIZATION*)
295 |  END.(*MAIN PROGRAM TERMINATED*)
296 |  (*READ 11A'S,B'S AND C'S AND 5 P'S*)
297 |  %EOF
Execution begins...
OUTSQR(1)0
OUTSQR(2)0
OUTSQR(3)0
OUTSQR(4)0
OUTSQR(5)0
OUTSQR(6)0
OUTSQR(7)0
OUTSQR(8)0
OUTSQR(9)0
OUTSQR(10)0
OUTSQR(11)0
OUTSQR(12)0
OUTSQR(13)0
OUTSQR(14)0
OUTSQR(15)0
OUTSQR(16)0
```

```
OUTSQR(18)0
OUTSQR(19)0
OUTSQR(20)0
OUTSQR(21)0
OUTSQR(22)0
OUTSQR(23)1
OUTSQR(24)0
OUTSQR(25)1
OUTSQR(26)0
OUTSQR(27)0
OUTSQR(28)0
OUTSQR(29)1

...execution ends
File 'EXSQST': 297 lines; no diagnostics
15113 bytes of object code generated
5550 statements executed
30016 bytes of memory requested during compilation
5792 bytes returned before execution
10392 bytes requested during execution
```

APPENDIX D

PROGRAM IMPLEMENTATION OF DATA-FLOW GRAPH GENERATION FROM THE EXPRESSION

```
1   | %$S 500000%
2   | (*                                                              *)
3   | (*                                                              *)
4   | (*              SPECIAL COMMENT ON THE FORMAT OF INPUT          *)
5   | (*                                                              *)
6   | (*     1) DOUBLE OPRAND INSTRUCTION MUST HAVE DOUBLE OPRANDS    *)
7   | (*        ENCLOSED IN THE BRACKET.(RECURSIVELY)                 *)
8   | (*     2) SINGLE OPRAND INSTRUCTIONS MUST HAVE SINGLE OPRANDS   *)
9   | (*        ENCLOSED IN THE BRACKETS.(RECURSIVELY).               *)
10  | (*     EXAMPLE:                                                 *)
11  | (*               (((¢4)+($4))+((2+4)*(3+2)))                   *)
12  | (*     SYMBOL ¢ DENOTES SQUARE AND $ DENOTES SQUARE-ROOT        *)
13  | (*     OPERATION.                                               *)
14  | (*                                                              *)
15  | (*                      O B J E C T I V E                       *)
16  | (*     THE PROGRAM TAKES THE EXPRESSION AS AN INPUT  AND        *)
17  | (*     GENERATES THE LEVEL-REPRESENTATION-MATRIX.THAT IS        *)
18  | (*     THE MATRIX REPRESENTS THE DATA-FLOW GRAPH OF  THE        *)
19  | (*     COMPUTATION EXPRESSION. THE OUTPUT IS THE REPRES-        *)
20  | (*     -ENTATION OF THE DATA-FLOW GRAPH.                        *)
21  | (*                                                              *)
22  | (*                                                              *)
23  | PROGRAM STACK(INPUT,OUTPUT);
24  | CONST
25  |    MAXARRYDIM=150;
26  | (*UPPER LIMIT ON THE NUMBER OF ICS ARE 150X150*)
27  | TYPE
28  |          (**((((((((((((((POINTER TYPE DEF))))))))))))))**)
29  |     LINK= @STACK;
30  |      STACK=
31  |        RECORD
32  |          NEXT:LINK;
33  |          DATA:CHAR;
34  |          ADDR1,ADDR2:INTEGER
35  |        END;
36  |  (*************************************************************)
37  |  (*  *DIMENSION OF LEVEL.MATRIX.REPRESENTATION ARRAY*         *)
38  |  SUB1=1..MAXARRYDIM;
39  |  (*************************************************************)
40  |  (**       SCALAR TYPE OPCODE DEFINATION WILL BE IN STAGE 2****)
41  |  (*                                                           *)
42  |  (*                    RECORED DEF OF L.R.M                   *)
43  |  (*         NOTICE IN THE L.R.M ,OPCODE WILL BE               *)
44  |  (*         PUT AS CHARACTER TO BE READ BY STAGE              *)
45  |  (*                        2                                  *)
46  |  (*         THUS STAGE 2 WILL HAVE SCALAR TYPE OPCODE         *)
47  |  (*                      DEFINATION                           *)
48  |       LRMAT=
49  |          RECORD
50  |           ADDRESS1,ADDRESS2,ADDOPR1,ADDOPR2:INTEGER;
51  |           DIROPR1,DIROPR2,OPCODE:CHAR
52  |          END;
53  | VAR
54  |       B,P,BASE1,BASE2,Q,TEMPPTR:LINK;
55  |       CH,TEMPCH:CHAR;
56  |       II,TEMPINT1,TEMPINT2,COUNT,COUNT1,COUNT2:INTEGER;
57  |       LRMATRIX:ARRAY (.SUB1,SUB1.) OF LRMAT;
58  |       (*                                                      *)
59  |       (*                TO BE USED BY SUB. SCANWRITE          *)
60  |       (***********COLOUMN LOCATOR IS DEFINED AS A MATRIX*******)
```

```
65  |       PTR@.DATA:=DATUM;
66  |       PTR@.ADDR1:=INT1;
67  |       PTR@.ADDR2:=INT2;
68  |       PTR@.NEXT:=BASE;
69  |       BASE:=PTR
70  |     END;(*END PROCEDURE PUSH*)
71  |   (*NOTICE THAT THE PROCEDURE WILL RETURN THE VALUE OF PTR AND BASE*)
72  |   (*************************************************************)
73  |   (* PROCEDURE POPS THE ELEMENTS I FROM THE TOP OF THE STACK*)
74  |   (*AGAIN NOTICE THAT THE VARIABLE PTR WILL RETURN THE VALUE OF POINTER*)
75  |   PROCEDURE POP (VAR PTR :LINK;I:INTEGER);
76  |   LABEL 4;
77  |    VAR
78  |        II:INTEGER;
79  |    BEGIN
80  |          FOR II:=1 TO I DO
81  |           BEGIN
82  |            PTR:=PTR@.NEXT;
83  |            IF (PTR=NIL)THEN
84  |                        BEGIN
85  |                          WRITELN('THIS STACK IS EMPTY NOW':20);
86  |                          GOTO 4
87  |                        END;
88  |          END; 4:
89  |    END;(*PROCEDURE*)
90  |   (*************************************************************)
91  |   PROCEDURE RITE(BC:LINK);
92  |     BEGIN
93  |          WHILE (BC<> NIL) DO
94  |             BEGIN(*WHILE*)
95  |               WRITE('********':4,BC@.DATA:2,'*********':4);
96  |               WRITELN(BC@.ADDR1:2,':':2,BC@.ADDR2:2);
97  |               BC:=BC@.NEXT
98  |             END;(*WHILE*)
99  |          WRITELN;
100 |          WRITELN
101 |     END;(*PROCEDURE RITE*)
102 |   (*                                                           *)
103 |   (*                                                           *)
104 |   (*                    PROCEDURE WRITMATRIX                   *)
105 |   (*************************************************************)
106 |   PROCEDURE WRITMATRIX(II,JJ:INTEGER);
107 |    BEGIN(*PROC*)
108 |     WRITELN('!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!':30);
109 |     WRITELN('LEVELNO':10,II:3,'COLOUMN NO':15,JJ:3);
110 |     WRITELN('OPCODE 1':8);
111 |     WRITELN('LEVELNO':8,LRMATRIX(.II,JJ.).ADDRESS1:3,'COLOUMN NO':15,
112 |     LRMATRIX(.II,JJ.).ADDOPR1:3,'OPRAND VALUE':20,LRMATRIX(.II,JJ.).DIROPR1:3);
113 |     WRITELN('OPCODE 2':8);
114 |     WRITELN('LEVELNO':8,LRMATRIX(.II,JJ.).ADDRESS2:3,'COLOUMN NO':15,
115 |     LRMATRIX(.II,JJ.).ADDOPR2:3,'OPRAND VALUE':20,LRMATRIX(.II,JJ.).DIROPR2:3);
116 |     WRITELN('OPERATION':10,LRMATRIX(.II,JJ.).OPCODE:3);
117 |     WRITELN('!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!':30)
118 |    END;(*PROC*)
119 |   (*                                                           *)
120 |   (*                  PROCEDURE SCANWRITE HERE                 *)
121 |   (*                                                           *)
122 |   PROCEDURE SCANWRITE(VAR PTR,BASE:LINK);
123 |   (*                                                           *)
124 |   (*              LOCAL VARIABLES                              *)
125 |    VAR
```

```
128 |      TEMPDATA:CHAR;
129 |   (*                                                              *)
130 |   (*                                                              *)
131 |   BEGIN (*PROCEDURE*)
132 |   (*                                                              *)
133 |   (*                                                              *)
134 |       BEGIN (*TRANSVERSE*)
135 |   (*                                                              *)
136 |   (*BY TRANSVERSING THE ELEMENTS TRACE APPROPRIATE LEVEL AND      *)
137 |   (*DISTUINGUISH THE PACKET OF OPCODE AND OPRAND SET              *)
138 |   (*                                                              *)
139 |   (**                                                             *)
140 |         MAXLEVEL:=0;
141 |         B:=PTR;
142 |         IF((B@.DATA)<>'(')THEN
143 |          WRITELN('ERROR 1 IN POINTERS IN PROCEDURE SCANWRITE':25);
144 |         NOOFPOP:=1;
145 |            WHILE ((B@.DATA)<>')') DO
146 |   (*                                                              *)
147 |   (*       IMPORTANT: IT IS ASSUMED THAT ')' AND '(' WILL HAVE    *)
148 |   (*       ADDR1 AND ADDR2 FIELD IN STACK AS 0:0                  *)
149 |   (*                                                              *)
150 |              BEGIN (*WHILE*)
151 |                   IF((B@.ADDR1)>MAXLEVEL)THEN MAXLEVEL:=B@.ADDR1;
152 |                   B:=B@.NEXT;
153 |                   NOOFPOP:=NOOFPOP+1
154 |              END;(*WHILE*)
155 |       IF ((B@.DATA)<> ')')THEN
156 |         WRITELN('ERROR 2 IN POINTER PROCEDURE SCANWRITE':30);
157 |   (*                                                              *)
158 |   (*       NOW SETTING THE APPROPRIATE LEVELS                     *)
159 |         IF (MAXLEVEL=0)THEN
160 |            LEVELINDX:=1(*IE. WE NEED TO HAVE LEVEL 1*)
161 |          ELSE
162 |            LEVELINDX:=MAXLEVEL +1;
163 |       END;(*TRANSVERSE*)
164 |   (*    NOOFPOP INDICATES THE NOOF POPS TO BE DONE,LEVEL INDEX  *)
165 |   (*    ALSO FOUND.                                             *)
166 |   (*                                                            *)
167 |   (*                                                            *)
168 |         BEGIN(*ADDRESS WRITTING ON STACK,POP*)
169 |   (*                                                            *)
170 |   (*              I M P O R T A N T                             *)
171 |   (*       INFORMATION FOR STAGE 2:*)
172 |   (*       1)((4)/(2)) IS WRONG                                   *)
173 |   (*       2)SINGLE OPRAND INST.($4)
174 |   (*       3)SQRT(2)=> ($2)*)
175 |   (*       4)SQR(2)=> (¢2)*)
176 |   (*       5)((4+2)) IS WRONG .NO TWO BRACKETS UNLESS OPERATION *)
177 |   (*         ENCLOSED.                                          *)
178 |   (*                                                            *)
179 |   (*                                                            *)
180 |           IF(PTR@.NEXT@.DATA) IN ('.','$','¢'.) THEN
181 |       BEGIN(*A :SINGLE OPRAND INST.*)
182 |                 IF((PTR@.NEXT@.NEXT@.NEXT@.DATA)<>')') THEN
183 |    WRITELN('WRONG FORMAT OR POINTER IN SINGLE OPRAND INST.':40)
184 |                ELSE
185 |           BEGIN(*A1*)
186 |                     LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).OPCODE:=
187 |                     PTR@.NEXT@.DATA;
188 |   (*                                                            *)
189 |   (*                NOW THE ADDRESS                             *)
190 |                     IF((PTR@.NEXT@.NEXT@.DATA)='0')THEN
```

```
193 |                          ADDRESS1:=PTR@.NEXT@.NEXT@.ADDR1;
194 |                          LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).
195 |                          ADDRESS2:=0;
196 |                          LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).
197 |                          ADDOPR2:=0;
198 |                          LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).
199 |                          ADDOPR1:=PTR@.NEXT@.NEXT@.ADDR2;
200 |                          LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).
201 |                          DIROPR1:='0';
202 |                          LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).
203 |                          DIROPR2:='0';
204 |                          WRITMATRIX(LEVELINDX,COLMNLOC(.LEVELINDX.))
205 |                          (*CALL PROCDEURE WRITE MATRIX*)
206 |                         END(*1*)
207 |                        ELSE
208 |                         BEGIN(*NOT 1*)
209 |   IF(((PTR@.NEXT@.NEXT@.ADDR1)<>0)AND((PTR@.NEXT@.NEXT@.ADDR2)<>0))THEN
210 |   WRITELN('STACK IS NOT INITILIZED CORRECTLY,I,E DIRECT OPRAND =>ADDRF: 0:0':30);
211 |                           LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).
212 |                           ADDRESS1:=0;
213 |                           LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).
214 |                           ADDOPR1:=0;
215 |                           LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).
216 |                           DIROPR1:=PTR@.NEXT@.NEXT@.DATA;
217 |                           LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).
218 |                           DIROPR2:='0';
219 |                           LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).
220 |                           ADDRESS2:=0;
221 |                           LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).
222 |                           ADDOPR2:=0;
223 |   (*                                                                  *)
224 |   (*        INFORMATION FOR SECOND STAGE:                             *)
225 |   (*        1)ADDR FIELD 0:0:0:0 INDICATES THAT A CONSTANT            *)
226 |   (*          OPRAND FOR SING.OPR IS IN DIROPR1 AND NOT DIROPR2       *)
227 |   (*                                                                  *)
228 |   IF (LEVELINDX<>1)THEN
229 |   WRITELN('TRACE IS NOT WORKING RIGHT,HERE LEVINDX SHOULD BE 1':40);
230 |                           WRITMATRIX(LEVELINDX,COLMNLOC(.LEVELINDX.))
231 |                          END;(*NOT 1*)
232 |   (*                                                                  *)
233 |   (*                POPPING AND WRITTING ON THE STACK IS DONE         *)
234 |                     POP(PTR,NOOFPOP);(*POP ALL ELEMENT*)
235 |   (*                                                                  *)
236 |   (*                 NOW, PUSHING DOWN THE ADDRESS                    *)
237 |                       TEMPADD1:=LEVELINDX;
238 |                       TEMPADD2:=COLMNLOC(.LEVELINDX.);
239 |                       TEMPDATA:='0';
240 |                       BASE:=PTR;
241 |                       NEW(PTR);
242 |                       PUSH(PTR,BASE,TEMPDATA,TEMPADD1,TEMPADD2);
243 |   (*                                                                  *)
244 |   (*       UPDATING THE COLUMN FOR THE NEXT TIME                      *)
245 |                       COLMNLOC(.LEVELINDX.):=COLMNLOC(.LEVELINDX.)+1
246 |             END;(*A1*)
247 |         END(*A*)
248 |   (*                                                                  *)
249 |   (*                 ELSE NOW DOUBLE OPRAND INSTRUCTION               *)
250 |    ELSE
251 |        BEGIN(*B"EG (2+5)*)
252 |           IF NOT((PTR@.NEXT@.NEXT@.DATA)IN(.'-','+','*','/'.))THEN
253 |   WRITELN('WRONG FORMAT OR POINTER IN DOUBLE OPRAND INSTRUCTION':60);
254 |   IF(PTR@.NEXT@.NEXT@.NEXT@.DATA<>')')THEN
255 |    WRITELN('POINTER OR FORMAT ERROR IN DOUBLE OPR.INS':60)
```

```
258   (*                    OK    FORMAT                                  *)
259    BEGIN(*B1*)
260      LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).OPCODE:=
261      PTR@.NEXT@.NEXT@.DATA;
262   (*NOW THE ADDRESS*)
263      IF ((PTR@.NEXT@.DATA)='0')THEN
264        BEGIN (*B11*)
265          LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).ADDRESS1:=
266          PTR@.NEXT@.ADDR1;
267          LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).ADDOPR1:=
268          PTR@.NEXT@.ADDR2;
269          LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).DIROPR1:='0'
270        END(*B11*)
271      ELSE
272   (*                                                                  *)
273        BEGIN(*NOT B11*)
274   IF(((PTR@.NEXT@.ADDR1)<>0)AND((PTR@.NEXT@.ADDR2)<>0))THEN
275   WRITELN('AFTER POPING, PROCEDURE SCANWRITE IS NOT WRITTING CORRECTLY ONSTACK'
276   :45);
277          LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).ADDRESS1:=0;
278          LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).ADDOPR1:=0;
279          LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).DIROPR1:=
280          PTR@.NEXT@.DATA
281        END;(*NOT B11*)
282   (*                                                                  *)
283   (*        GOING FOR THE NEXT OPRAND AND DOING SAME                  *)
284      IF ((PTR@.NEXT@.NEXT@.NEXT@.DATA)='0')THEN
285       BEGIN (*B12*)
286          LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).ADDRESS2:=
287          PTR@.NEXT@.NEXT@.NEXT@.ADDR1;
288          LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).ADDOPR2:=
289          PTR@.NEXT@.NEXT@.NEXT@.ADDR2;
290          LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).DIROPR2:='0'
291        END(*B12*)
292      ELSE
293   (*                                                                  *)
294        BEGIN(*NOT B12*)
295   IF(((PTR@.NEXT@.NEXT@.NEXT@.ADDR1)<>0)AND((PTR@.NEXT@.NEXT@.NEXT@.ADDR2 )<>0))
296   THEN
297   WRITELN('AFTER POPING, PROCEDURE SCANWRITE IS NOT WRITTING CORRECTLY ONSTACK'
298   :45);
299          LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).ADDRESS2:=0;
300          LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).ADDOPR2:=0;
301          LRMATRIX(.LEVELINDX,COLMNLOC(.LEVELINDX.).).DIROPR2:=
302          PTR@.NEXT@.NEXT@.DATA
303        END;(*NOT B12*)
304        WRITMATRIX(LEVELINDX,COLMNLOC(.LEVELINDX.));
305   (*                                                                  *)
306   (*  POPPING AND WRITTING ON THE STACK IS DONE                      *)
307         POP(PTR,NOOFPOP);
308   (*                                                                  *)
309   (*        PUSHING DOWN THE ADDRESS                                    *)
310        TEMPADD1:=LEVELINDX;
311        TEMPADD2:=COLMNLOC(.LEVELINDX.);
312        TEMPDATA:='0';
313        BASE:=PTR;
314        NEW(PTR);
315        PUSH(PTR,BASE,TEMPDATA,TEMPADD1,TEMPADD2);
316        COLMNLOC(.LEVELINDX.):=COLMNLOC(.LEVELINDX.)+1
317      END;(*B1*)
318     END;(*B*)
319    END;(*ADDRESS WRITTING ON STACK*)
320  END;(*PROCEDURE*)
```

```
(*                                                                    *)
(*                                                                    *)
(*************** M A I N   P R O G R A M ********************)
BEGIN(*MAIN PG*)
(*                                                                    *)
(*                  INITILIZATION                                     *)
(*                                                                    *)
        COUNT:=0;COUNT1:=0;COUNT2:=0;
        BASE1:=NIL;BASE2:=NIL;
(*                                                                    *)
(*              INITILIZING THE COLMNLOCATOR TO 1                     *)
        FOR II:=1 TO MAXARRYDIM DO
                COLMNLOC(.II.):=1;
(*                                                                    *)
(*              INITILIZATION DONE                                    *)
(*                                                                    *)
(**********READING IN THE STACK VALUE FROM UP*******)
        WHILE NOT EOF DO
              BEGIN
                NEW(P);
                READ(CH);
                IF(CH='(')THEN COUNT1:=COUNT1+1;
                IF(CH=')')THEN COUNT2:=COUNT2+1;
                PUSH(P,BASE1,CH,0,0);(*STORING ADDRESS FIELDS OF 0*)
                COUNT:=COUNT+1
        END;(*WHILE*)
 IF(COUNT1=COUNT2) THEN
   BEGIN(*OK*)
        POP(P,1);(*MAKING THE WHILE EOF CATER FOR*)
        COUNT:=COUNT-1;(*COUNT CARRIES THE COUNT OF # OF CHAR ON STACK*)
        RITE(P);
        WRITELN('THIS WAS THE STACK JUST READ':50);
        (*************************************************************)
        (*           MAKING THE STACK CIRCULAR NOW                   *)
        REPEAT
            BEGIN(*CIRCLE*)
(*                                                                    *)
(*                                                                    *)
(*              READING IN FROM STACK UP                              *)
(*                                                                    *)
                TEMPCH:=P@.DATA;
                TEMPINT1:=P@.ADDR1;
                TEMPINT2:=P@.ADDR2;
(*                                                                    *)
(*              READING COMPLETE FROM THE STACK UP                    *)
                POP(P,1);
                NEW(Q);
                PUSH(Q,BASE2,TEMPCH,TEMPINT1,TEMPINT2);
(*                                                                    *)
(*                                                                    *)
(*              CALLING THE SCANWRITE NOW                             *)
(*                                                                    *)
(*      REMEMBER THAT THE VALUE OF Q AND BASE2 WILL BE CHANGED        *)
(*      BY SCANWRITE LIKE PROC.POP AND PUSH IE VAR PTR,BASE           *)
(*      USED IN THE PROCEDURE OF SCANWRITE                            *)
(*                                                                    *)
(*                                                                    *)
                IF (TEMPCH='(')THEN
                  SCANWRITE(Q,BASE2);
            END(*CIRCLE*)
        UNTIL (P=NIL)
   END(*LOOP*)
 ELSE(*NOT OK*)
```

```
  388  |  %EOF
Execution begins...
**** )**** 0 : 0
**** )**** 0 : 0
**** 1**** 0 : 0
**** ***** 0 : 0
**** 2**** 0 : 0
**** (**** 0 : 0
**** +**** 0 : 0
**** )**** 0 : 0
**** )**** 0 : 0
**** 2**** 0 : 0
**** -**** 0 : 0
**** 3**** 0 : 0
**** (**** 0 : 0
**** ***** 0 : 0
**** )**** 0 : 0
**** 4**** 0 : 0
**** +**** 0 : 0
**** 5**** 0 : 0
**** (**** 0 : 0
**** (**** 0 : 0
**** (**** 0 : 0


                        THIS WAS THE STACK JUST READ
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   LEVELNO  1     COLOUMN NO  1
OPCODE 1
 LEVELNO  0      COLOUMN NO  0         OPRAND VALUE  2
OPCODE 2
 LEVELNO  0      COLOUMN NO  0         OPRAND VALUE  1
 OPERATION  *
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   LEVELNO  1      COLOUMN NO  2
OPCODE 1
 LEVELNO  0      COLOUMN NO  0         OPRAND VALUE  3
OPCODE 2
 LEVELNO  0      COLOUMN NO  0         OPRAND VALUE  2
 OPERATION  -
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   LEVELNO  1      COLOUMN NO  3
OPCODE 1
 LEVELNO  0      COLOUMN NO  0         OPRAND VALUE  5
OPCODE 2
 LEVELNO  0      COLOUMN NO  0         OPRAND VALUE  4
 OPERATION  +
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   LEVELNO  2      COLOUMN NO  1
OPCODE 1
 LEVELNO  1      COLOUMN NO  3         OPRAND VALUE  0
OPCODE 2
 LEVELNO  1      COLOUMN NO  2         OPRAND VALUE  0
 OPERATION  *
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
THIS STACK IS EMPTY
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   LEVELNO  3      COLOUMN NO  1
OPCODE 1
 LEVELNO  2      COLOUMN NO  1         OPRAND VALUE  0
```

```
 LEVELNO  1      COLOUMN NO  1          OPRAND VALUE  0
 OPERATION  +
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
THIS STACK IS EMPTY

...execution ends
File 'ARYFORM': 388 lines; no diagnostics
15655 bytes of object code generated
1445 statements executed
32880 bytes of memory requested during compilation
5792 bytes returned before execution
501928 bytes requested during execution
```

APPENDIX E

PROGRAM IMPLEMENTATION OF INSTRUCTION GENERATION FROM THE DATA-FLOW GRAPH REPRESENTATION (with optimality consideration only)

```
PROGRAM GENERATEIC( INPUT,OUTPUT);
(*                                                                  *)
(*                                                                  *)
(*                        OBJECTIVE                                 *)
(*     ONCE THE STAGE1 OF OPTCOMPILER GIVES LEVEL REPERS-           *)
(*    -ENTATION MATRIX WHICH EXACTLY REPRESENTS THE                 *)
(*     THE DATA-FLOW GRAPH, THIS ALGORITHM GENERATES THE            *)
(*     INSTRUCTION CELLS TO BE LOADED INTO MEMORY OF HOST           *)
(*     COMPUTER. THE HOST COMPUTER TAKES THE IC ARRAY AND MAPS      *)
(*     IT SO THAT THE Q U E U E   I N  D O W N  L O A D I N G       *)
(*     TAKES PLACE FROM   THE OUTPUT PORT OF THE HOST INTO THE      *)
(*     DFEE SYSTEM IN THE MANNER SHOWN IN THE ARCHITECTURE.         *)
(*                                                                  *)
(*                                                                  *)
(*                      P L E A S E  N O T E                        *)
(*     THIS OPTIMIZATION COMPILER DOES NOT TAKE CARE OF THE         *)
(*     LOOP SCHEMA AND IT WOULD NOT BE IMPLEMENTED AND SINCE        *)
(*     TWO DATA-FLOW GRAPH ARE USED,TWO SUCH LRM SHOULD BE FORMED   *)
(*     AND THEN CONFILICT WOULD BE CHECKED. AGAIN THIS WILL ONLY    *)
(*     BE DESCRIBED AS ALGORITHM AND NOT IMPLEMENTED.               *)
(*     THIS PROGRAM IMPLEMENTATION TAKES CARE OF ONLY OPTIMAL       *)
(*     PLACEMENT OF INSTRUCTIONS IN THE INSTRUCTION CELLS.          *)
(*                                                                  *)
(*                                                                  *)
(*                                                                  *)
(*                   C O N F L I C T   CHECKING                     *)
(*     BEFORE THE ACTUAL LOADING TAKES PLACE, THE PROCEDURE         *)
(*     CONFLICT IS  C A L L E D  WHICH SCANS FOR CONFLICTS          *)
(*     MENTIONED IN ARCHITECTRAL DESCRIPTION AND SWAPS              *)
(*     THE CONFLICTED CELL WITH A BLANK CELL FROM BELOW.            *)
(*     IF NO BALNK CELL EXISTS, THE OPTICOMPILER GIVES AN           *)
(*     ERROR MESSAGE OF CONFLICT OCCURING AND ANOTHER FORM          *)
(*     OF EXPRESSION SHOULD BE TRIED ON.                            *)
(*     HOWEVER, NOTE THAT THE PROCEDURE  C O N F L I C T            *)
(*     WILL NOT BE IMPLEMENTED AND JUST ALGORITHM  WILL BE          *)
(*     GIVEN FOR IT.                                                *)
(*                                                                  *)
(*                                                                  *)
(*                                                                  *)
(*      NOTICE THAT NULL INSERTION IS DONE WHEN THE                 *)
(*    ADDRESSING TO OTHER LEVELS (J+1) .. FROM LEVEL J              *)
(*                    IS MORE THAN TWICE.                           *)
(*    HOWEVER, THIS WILL BE DESCRIBED IN THE ALGORITHM              *)
(*    AND NOT IMPLEMENTED                                           *)

(*                      INPUT TO THIS PROCEDURE                     *)
(*                                                                  *)
(*                 1)MAXIMUM NO OF ICS.(MAXIC)                      *)
(*                 2)MAXIMUM NO OF CBS.(MAXCB)                      *)
(*                 3)L.R.M FROM STAGE1                              *)
(*                                                                  *)
(*                                                                  *)
CONST
     MAXCELBLK=2;
     MAXIC=4;
     MAXADDRESS=2;
     TEMPMAXCELBLK=8;(*THIS SHOULD BE EQUAL TO LEVELINDEX OR MORE*)
(*                                                                  *)
(*                                                                  *)
(*                                                                    *)
```

```
(*                  FROM LEVEL.REP. MATRIX                          *)
(*            C R I T I C A L   I N F O R M A T I O N               *)
(*                                                                  *)
    MAXCOLMN=3; (*TO BE USED FOR DECLARING ARRAY*)
    LEVELINDEX=3;
(*              MAX(3,2,1)=3                                        *)
    MAXMULT=9;(*MAXCOLMN X LEVELINDEX*)
(*    THIS WILL BE USED BY CHANGEADDRESS PROCEDURE                  *)
(*                                                                  *)
(*                                                                  *)
(*                                                                  *)
TYPE
(*                                                                  *)
(*      NOTICE HERE WE NEED TO STORE ALSO HOW MANY PLACED IS        *)
(*      IS THE RESULT NEED.SO COUNT FIELD IS NEEDED AND AN ARRAY*)
(*      OF MAXIMIUM OF 10 ELEMENTS TO REPRESENT THE 10 ADDRESSES*)
(*      IN THE MAXIMIUM CASE, THE COUNT SHOULD BE 10.               *)
(*                                                                  *)
(*      DEFINING THESE 10 ADDRESS AS 1-D ARRAY                      *)
(*                                                                  *)
(*                                                                  *)
(*                                                                  *)
         ADDR=
          RECORD
           LEVELIND:INTEGER;
           COLMNIND:INTEGER;
           OPRIND:INTEGER
          END;
         MAT1=ARRAY(.1..MAXADDRESS.) OF ADDR;
(*                                                                  *)
(*      NEXT DEFINING THE MODIFIED LRMATRIX                         *)
(*      THIS IS TO BE GIVEN WITHOUT THE INFO OF COUNT               *)
(*      AND ADDCOUNT. BUT HERE, IT IS READ FROM ANINPUT FILE        *)
(*                                                                  *)
(*                                                                  *)
         LRMAT=
              RECORD
               ADDRESS1,ADDRESS2,ADDOPR1,ADDOPR2,COUNT:INTEGER;
               DIROPR1,DIROPR2,OPCODE:CHAR;
               ADDCOUNT:MAT1
              END;
(*                                                                  *)
(*                                                                  *)
(*                                                                  *)
(*    NOW DEFINING THE OUTPUT OF THE PROGRAM I.E                    *)
(*    THE INSTRCELL.THIS IS AN ARRAY OF RECORD . THE SIZE OF        *)
(*    THE ARRAY IS EQUAL TO A 2-D (MAXCEKBLK,MAXIC).I,E FIXED       *)
(*    THIS OUTPUT CAN DIRECTLY BE USED IN THE SIMULATION PORGRAM*)
(*    I.E THE OUTPUT OF THIS PROGRAM IS DIRECTLY COMPATABLE         *)
(*    WITH THE SIMULATION ARCHITECTURE.                             *)
(*                                                                  *)
(*                                                                  *)
 NUMCELBLK=0..TEMPMAXCELBLK;
 NUMIC=0..MAXIC;
 NUMOPR=0..2;
 STARVAL='A'..'D';
 (*                                                                 *)
 (*        NOTICE THAT WE WILL BE ONLY USING  'A' AND 'B'.          *)
 ADDSPEC=
   RECORD
     CB:NUMCELBLK;
     IC:NUMIC;
     OPR:NUMOPR
```

```
125 |    INSTRCELL=
126 |      RECORD
127 |        OPCODE:CHAR;
128 |        OPRAND1,OPRAND2,RESULT:CHAR;(*IN ORDER TO BE COMPATABLE WITH LRMATRIX*)
129 |        STAR:STARVAL;
130 |        ADDRSLT1,ADDRSLT2:ADDSPEC
131 |      END;
132 |   (*                                                                  *)
133 |   (*          REMEMBER THAT BLANK RESULT IS REPRESENTED AS            *)
134 |   (*                              ''                                  *)
135 |   (*                              ''                                  *)
136 |   (*                              ''                                  *)
137 |     CHANGE=
138 |       RECORD
139 |        CELNO,ICNO:INTEGER
140 |       END;
141 |   (*                              ''                                  *)
142 |   (*                                                                  *)
143 |    SUB1=1..TEMPMAXCELBLK;
144 |    SUB2=1..MAXIC;
145 |    SUB3=1..LEVELINDEX;
146 |    SUB4=1..MAXCOLMN;
147 |    SUB5=1..MAXMULT;
148 |   (*                                                                  *)
149 |   (*                                                                  *)
150 |     MAT2=ARRAY(.SUB1,SUB2.)OF INSTRCELL;
151 |     MAT3=ARRAY(.SUB3,SUB4.)OF LRMAT;
152 |     MAT4=ARRAY(.SUB5.)OF CHANGE;(*TO BE USED BY PROCEDURE CHANGEADDRESS*)
153 |   (*                                                                  *)
154 |   (*                                                                  *)
155 |   (*                                                                  *)
156 |   VAR
157 |
158 |   (*              OUTPUT VARIABLE                                     *)
159 |   (*                                                                  *)
160 |    CELLBLOCK:MAT2;
161 |   (*                                                                  *)
162 |   (*              INPUT VARIABLE                                      *)
163 |   (*                                                                  *)
164 |    LRMATRIX:MAT3;
165 |    COLMN:ARRAY(.SUB3.)OF INTEGER;
166 |    UPCOUNT1,UPCOUNT2:MAT4;
167 |   (*                                                                  *)
168 |    I,J,L,K,ICE,CBE,TEMP,STR1,STR2,INDX1,INDX2:INTEGER;
169 |   (*                                                                  *)
170 |   (*                                                                  *)
171 |   (*                        PROCEDURE MATRIX                          *)
172 |   PROCEDURE MATRIX(VAR CNT:INTEGER;OPRNO:INTEGER);
173 |    BEGIN
174 |           IF (CNT<=MAXADDRESS) THEN
175 |            BEGIN(*IF BEGIN*)
176 |              CNT:=CNT+1;
177 |              LRMATRIX(.I,J.).ADDCOUNT(.CNT.).LEVELIND:=L;
178 |              LRMATRIX(.I,J.).ADDCOUNT(.CNT.).COLMNIND:=K;
179 |              LRMATRIX(.I,J.).ADDCOUNT(.CNT.).OPRIND:=OPRNO
180 |            END;(*IF END*)
181 |          IF(LRMATRIX(.I,J.).COUNT>MAXADDRESS)THEN
182 |          WRITELN('EXPRESSION HAS MORE ADDRESSING PER OUTPUT,LIMIT EXCEEDED':50)
183 |    END;
184 |   (*                                                                  *)
185 |   (*                                                                  *)
186 |   PROCEDURE OPRD(GMATRIX:MAT3;I1,J1,CB1,IC1:INTEGER);
187 |      BEGIN
```

```
190 |                          IF((GMATRIX(.I1,J1.).ADDRESS1<>0)AND
191 |                           (GMATRIX(.I1,J1.).ADDOPR1<>0))THEN
192 |                             CELLBLOCK(.CB1,IC1.).OPRAND1:=' '
193 |                          ELSE
194 |    CELLBLOCK(.CB1,IC1.).OPRAND1:=GMATRIX(.I1,J1.).DIROPR1
195 |                        END(*A11*)
196 |                        ELSE(*NOT A11*)
197 |   (*                                                              *)
198 |   (*                        DOUBLE OPRAND INSTRUCTION             *)
199 |                        BEGIN(*A12*)
200 |                           IF((GMATRIX(.I1,J1.).ADDRESS1<>0)AND
201 |                           (GMATRIX(.I1,J1.).ADDOPR1<>0))THEN
202 |                             CELLBLOCK(.CB1,IC1.).OPRAND1:=' '
203 |                           ELSE
204 |                             CELLBLOCK(.CB1,IC1.).OPRAND1:=
205 |                             GMATRIX(.I1,J1.).DIROPR1;
206 |   (*                          SECOND OPRAND                       *)
207 |   (*                                                              *)
208 |                           IF((GMATRIX(.I1,J1.).ADDRESS2<>0)AND
209 |                           (GMATRIX(.I1,J1.).ADDOPR2<>0))THEN
210 |                             CELLBLOCK(.CB1,IC1.).OPRAND2:=' '
211 |                           ELSE
212 |                             CELLBLOCK(.CB1,IC1.).OPRAND2:=
213 |                             GMATRIX(.I1,J1.).DIROPR2
214 |                         END(*A12*)
215 |          END;
216 |   (*                                                              *)
217 |   (*                                                              *)
218 |   PROCEDURE INCREMENT(VAR ICB,IIC:INTEGER);
219 |    BEGIN
220 |         ICB:=ICB+1;
221 |         IF(ICB>MAXCELBLK)THEN
222 |          BEGIN
223 |           ICB:=1;
224 |           IIC:=IIC+1;
225 |   IF(IIC>MAXIC)THEN WRITELN('MORE INSTRUCTIONS THAN MEMORY':40)
226 |          END;
227 |    END;
228 |   (*                                                              *)
229 |   (*                                                              *)
230 |   PROCEDURE RITEM(CELL:MAT2;M1,M2:INTEGER);
231 |    BEGIN
232 |   WRITELN('************************************************************':60);
233 |        WRITELN('CELLBLOCK':40,M1:4,'!INST.NO':9,M2:2);
234 |        WRITELN('CELLBLOCK.OPCODE':40,CELL(.M1,M2.).OPCODE:5);
235 |        WRITELN('CELLBLOCK.OPRAND1':40,CELL(.M1,M2.).OPRAND1:3);
236 |        WRITELN('CELLBLOCK.OPRAND2':40,CELL(.M1,M2.).OPRAND2:3);
237 |        WRITELN('CELLBLOCK.RESULT':40,CELL(.M1,M2.).RESULT:4);
238 |        WRITELN('CELLBLOCK.STAR':40,CELL(.M1,M2.).STAR:4);
239 |        WRITELN('CELLBLOCK.ADDRSLT1.CB':40,CELL(.M1,M2.).ADDRSLT1.CB:3);
240 |        WRITELN('CELLBLOCK.ADDRSLT1.IC':40,CELL(.M1,M2.).ADDRSLT1.IC:3);
241 |        WRITELN('CELLBLOCK.ADDRSLT1.OPR':40,CELL(.M1,M2.).ADDRSLT1.OPR:3);
242 |        WRITELN('CELLBLOCK.ADDRSLT2.CB':40,CELL(.M1,M2.).ADDRSLT2.CB:3);
243 |        WRITELN('CELLBLOCK.ADDRSLT2.IC':40,CELL(.M1,M2.).ADDRSLT2.IC:3);
244 |        WRITELN('CELLBLOCK.ADDRSLT2.OPR':40,CELL(.M1,M2.).ADDRSLT2.OPR:3)
245 |     END;
246 |   (*                                                              *)
247 |   (*                                                              *)
248 |   PROCEDURE CHANGEADDRESS(VAR CHMATRIX:MAT3;II,JJ,CEL,IIN,MAX:INTEGER;
249 |                  VAR UPCNTA,UPCNTB:MAT4;VAR INDEX1,INDEX2:INTEGER);
250 |   VAR
251 |    VV,MM,JL:INTEGER;
252 |    VAL,OKGO1,OKGO2:BOOLEAN;
```

```
255     OR((CHMATRIX(.II,JJ.).ADDRESS2<>0)AND (CHMATRIX(.II,JJ.).ADDOPR2<>0)))
256     THEN
257      BEGIN(*A*)
258    (*                                                              *)
259    (*                                                              *)
260    (*                                                              *)
261    (*      HERE, I CANNOT HELP SEARCHING THE WHOLE MEMORY SPACE    *)
262    (*         TO CHANGE THE LEVELINDEX AND COLMNLOC TO UPDATED     *)
263    (*                  CELL BLOCK ADDRESS VALUE                    *)
264    (*                                                              *)
265    (*                                                              *)
266    (*                                                              *)
267        FOR VV:=1 TO MAXCELBLK DO(*SEARCH FOR THE ADDRESS*)
268          BEGIN(*B*)
269           FOR MM:=1 TO MAXIC DO
270            BEGIN(*C*)
271            IF((CELLBLOCK(.VV,MM.).ADDRSLT1.CB=II)AND
272            (CELLBLOCK(.VV,MM.).ADDRSLT1.IC=JJ)) THEN
273      BEGIN(*MOVING*)
274    (*  CHECKING WHETHER THE CELL HAS ALREADY BEEN UPDATED ADDRESS VISE*)
275       BEGIN(*LAST*)
276       OKGO1:=FALSE;
277       FOR JL:=1 TO MAX DO
278        BEGIN(*LOOP*)
279         VAL:=((UPCNTA(.JL.).CELNO=VV)AND(UPCNTA(.JL.).ICNO=MM));
280         OKGO1:=OKGO1 OR VAL
281        END;(*LOOP*)
282       END;(*LAST*)
283    (*  CHECKING WHETHER THE CELL HAS ALREADY BEEN UPDATED ADDRESS VISE*)
284             IF NOT OKGO1 THEN
285             BEGIN(*1*)
286              CELLBLOCK(.VV,MM.).ADDRSLT1.CB:=CEL;
287              CELLBLOCK(.VV,MM.).ADDRSLT1.IC:=IIN;
288              UPCNTA(.INDEX1.).CELNO:=VV;
289              UPCNTA(.INDEX1.).ICNO:=MM;
290              INDEX1:=INDEX1+1
291             END;(*1*)
292           END;(*MOVING*)
293          END;(*C*)
294         END;(*B*)
295    (*                                                              *)
296    (*                  FOR ADDRSLT2                                *)
297    (*                                                              *)
298       FOR VV:=1 TO MAXCELBLK DO
299         BEGIN(*B1*)
300            FOR MM:=1 TO MAXIC DO
301           BEGIN(*C*)
302            IF((CELLBLOCK(.VV,MM.).ADDRSLT2.CB=II)AND
303            (CELLBLOCK(.VV,MM.).ADDRSLT2.IC=JJ)) THEN
304             BEGIN(*MOVING*)
305    (*  CHECKING WHETHER THE CELL HAS ALREADY BEEN UPDATED ADDRESS VISE*)
306       BEGIN(*LAST*)
307       OKGO2:=FALSE;
308       FOR JL:=1 TO MAX DO
309        BEGIN(*LOOP*)
310         VAL:=((UPCNTB(.JL.).CELNO=VV)AND(UPCNTB(.JL.).ICNO=MM));
311         OKGO2:=OKGO2 OR VAL
312        END;(*LOOP*)
313       END;(*LAST*)
314    (*  CHECKING WHETHER THE CELL HAS ALREADY BEEN UPDATED ADDRESS VISE*)
315             IF NOT OKGO2 THEN
316             BEGIN(*1*)
317               CELLBLOCK(.VV,MM.).ADDRSLT2.CB:=CEL;
```

```
320 |              UPCNTB(.INDEX2.).ICNO:=MM;
321 |              INDEX2:=INDEX2+1
322 |              END;(*1*)
323 |            END;(*MOVING*)
324 |          END;(*C*)
325 |        END;(*B*)
326 |      END;(*A*)
327 |    END;(*PROC*)
328 |  (*                                                              )*
329 |  (*                                                              )*
330 |  (*******************M A I N   P R O G R A M************************)
331 |  BEGIN
332 |
333 |  (*                                                              *)
334 |  (*                                                              *)
335 |  (*                                                              *)
336 |  (*                 INITILIZING THE ADDRESS OF CELLBLOCK TO 0     *)
337 |  (*               FOR COMPARISION IN PROC. CHANGEADDRESS          *)
338 |    FOR I:=1 TO TEMPMAXCELBLK DO
339 |      BEGIN
340 |        FOR J:=1 TO MAXIC DO
341 |          BEGIN
342 |            CELLBLOCK(.I,J.).ADDRSLT1.CB:=0;
343 |            CELLBLOCK(.I,J.).ADDRSLT2.CB:=0;
344 |            CELLBLOCK(.I,J.).ADDRSLT1.IC:=0;
345 |            CELLBLOCK(.I,J.).ADDRSLT2.IC:=0;
346 |            CELLBLOCK(.I,J.).OPCODE:=' ';
347 |            CELLBLOCK(.I,J.).OPRAND1:=' ';
348 |            CELLBLOCK(.I,J.).OPRAND2:=' ';
349 |            CELLBLOCK(.I,J.).RESULT:=' ';
350 |            CELLBLOCK(.I,J.).STAR:='D';
351 |            CELLBLOCK(.I,J.).ADDRSLT1.OPR:=0;
352 |            CELLBLOCK(.I,J.).ADDRSLT2.OPR:=0
353 |          END;
354 |      END;
355 |      FOR I:=1 TO MAXMULT DO
356 |       BEGIN
357 |        UPCOUNT1(.I.).CELNO:=0;
358 |        UPCOUNT2(.I.).CELNO:=0;
359 |        UPCOUNT1(.I.).ICNO:=0;
360 |        UPCOUNT2(.I.).ICNO:=0
361 |       END;
362 |       INDX1:=1;INDX2:=1;
363 |  (*          THESE ARE TO BE USED BY PROCEDURE CHANGE ADDRESS     *)
364 |  (*          TO MAKE SURE THAT                                    *)
365 |  (*                                                               *)
366 |  (*                                                               *)
367 |  (*                                                               *)
368 |  (*              INPUT SECTION BEGINS HERE                        *)
369 |  (*                FROM LEVEL.REP. MATRIX                        *)
370 |  (*                                                              *)
371 |  (*            C R I T I C A L   I N F O R M A T I O N           *)
372 |  (*                                                              *)
373 |      COLMN(.1.):=3;(*LEVELINDEX=1*)
374 |      COLMN(.2.):=2;(*LEVELINDEX=2*)
375 |      COLMN(.3.):=1;  (*LEVELINDEX=3*)
376 |  (*                                                              *)
377 |  (*                                                               *)
378 |  (*    THE EXPRESSION FOR THIS LRMATRIX IS:*)
379 |  (*    (((3-2)*(5+4))+(2*1))                                      *)
380 |              LRMATRIX(.1,1.).ADDRESS1:=0;
381 |              LRMATRIX(.1,1.).ADDRESS2:=0;
382 |              LRMATRIX(.1,1.).ADDOPR1:=0;
```

```
                    LRMATRIX(.1,1.).DIROPR2:='1';
                    LRMATRIX(.1,1.).OPCODE:='+';
                    LRMATRIX(.1,1.).COUNT:=0;
                     FOR I:=1 TO MAXADDRESS DO
                       BEGIN
                         LRMATRIX(.1,1.).ADDCOUNT(.I.).LEVELIND:=0;
                         LRMATRIX(.1,1.).ADDCOUNT(.I.).COLMNIND:=0;
                         LRMATRIX(.1,1.).ADDCOUNT(.I.).OPRIND:=0
                       END;
  (*                                                                  *)
                    LRMATRIX(.1,2.).ADDRESS1:=0;
                    LRMATRIX(.1,2.).ADDRESS2:=0;
                    LRMATRIX(.1,2.).ADDOPR1:=0;
                    LRMATRIX(.1,2.).ADDOPR2:=0;
                    LRMATRIX(.1,2.).DIROPR1:='2';
                    LRMATRIX(.1,2.).DIROPR2:='1';
                    LRMATRIX(.1,2.).OPCODE:='*';
                    LRMATRIX(.1,2.).COUNT:=0;
                     FOR I:=1 TO MAXADDRESS DO
                       BEGIN
                         LRMATRIX(.1,2.).ADDCOUNT(.I.).LEVELIND:=0;
                         LRMATRIX(.1,2.).ADDCOUNT(.I.).COLMNIND:=0;
                         LRMATRIX(.1,2.).ADDCOUNT(.I.).OPRIND:=0
                       END;
  (*                                                                  *)
                    LRMATRIX(.1,3.).ADDRESS1:=0;
                    LRMATRIX(.1,3.).ADDRESS2:=0;
                    LRMATRIX(.1,3.).ADDOPR1:=0;
                    LRMATRIX(.1,3.).ADDOPR2:=0;
                    LRMATRIX(.1,3.).DIROPR1:='3';
                    LRMATRIX(.1,3.).DIROPR2:='2';
                    LRMATRIX(.1,3.).OPCODE:='-';
                    LRMATRIX(.1,3.).COUNT:=0;
                     FOR I:=1 TO MAXADDRESS DO
                       BEGIN
                         LRMATRIX(.1,3.).ADDCOUNT(.I.).LEVELIND:=0;
                         LRMATRIX(.1,3.).ADDCOUNT(.I.).COLMNIND:=0;
                         LRMATRIX(.1,3.).ADDCOUNT(.I.).OPRIND:=0
                       END;
  (*                                                                  *)
  (*                                                                  *)
                    LRMATRIX(.2,1.).ADDRESS1:=1;
                    LRMATRIX(.2,1.).ADDRESS2:=1;
                    LRMATRIX(.2,1.).ADDOPR1:=1;
                    LRMATRIX(.2,1.).ADDOPR2:=2;
                    LRMATRIX(.2,1.).DIROPR1:='0';
                    LRMATRIX(.2,1.).DIROPR2:='0';
                    LRMATRIX(.2,1.).OPCODE:='-';
                    LRMATRIX(.2,1.).COUNT:=0;
                     FOR I:=1 TO MAXADDRESS DO
                       BEGIN
                         LRMATRIX(.2,1.).ADDCOUNT(.I.).LEVELIND:=0;
                         LRMATRIX(.2,1.).ADDCOUNT(.I.).COLMNIND:=0;
                         LRMATRIX(.2,1.).ADDCOUNT(.I.).OPRIND:=0
                       END;
  (*                                                                  *)
  (*                                                                  *)
                    LRMATRIX(.2,2.).ADDRESS1:=1;
                    LRMATRIX(.2,2.).ADDRESS2:=1;
                    LRMATRIX(.2,2.).ADDOPR1:=2;
                    LRMATRIX(.2,2.).ADDOPR2:=3;
                    LRMATRIX(.2,2.).DIROPR1:='0';
                    LRMATRIX(.2,2.).DIROPR2:='0';
```

```
                    FOR I:=1 TO MAXADDRESS DO
                      BEGIN
                        LRMATRIX(.2,2.).ADDCOUNT(.I.).LEVELIND:=0;
                        LRMATRIX(.2,2.).ADDCOUNT(.I.).COLMNIND:=0;
                        LRMATRIX(.2,2.).ADDCOUNT(.I.).OPRIND:=0
                      END;
  (*                                                                  *)
  (*                                                                  *)
                LRMATRIX(.3,1.).ADDRESS1:=2;
                LRMATRIX(.3,1.).ADDRESS2:=2;
                LRMATRIX(.3,1.).ADDOPR1:=1;
                LRMATRIX(.3,1.).ADDOPR2:=2;
                LRMATRIX(.3,1.).DIROPR1:='0';
                LRMATRIX(.3,1.).DIROPR2:='0';
                LRMATRIX(.3,1.).OPCODE:='/';
                LRMATRIX(.3,1.).COUNT:=0;
                 FOR I:=1 TO MAXADDRESS DO
                   BEGIN
                     LRMATRIX(.3,1.).ADDCOUNT(.I.).LEVELIND:=0;
                     LRMATRIX(.3,1.).ADDCOUNT(.I.).COLMNIND:=0;
                     LRMATRIX(.3,1.).ADDCOUNT(.I.).OPRIND:=0
                   END;
  (*                                                                  *)
  (*                                                                  *)
  (*                INPUT SECTION FROM L.R.MATRIX FINISHED            *)
  (*                                                                  *)
  (*                                                                  *)

  (*                                                                  *)
  (*                                                                  *)
  (*         COUNTING THE NUMBER OF INSTRUCTIONS WHICH HAVE           *)
  (*                BEEN ACCESSED NUMBER OF TIMES AND                 *)
  (*                    STORING IT IN COUNT                           *)
                  FOR I:=1 TO LEVELINDEX DO
                   BEGIN(*11*)
                    FOR J:=1 TO COLMN(.I.) DO
                      BEGIN(*12*)
                       FOR L:=(I+1) TO LEVELINDEX DO
                        BEGIN(*13*)
                         FOR K:=1 TO COLMN(.L.) DO
                          BEGIN(*14*)
  (*                                                                  *)
  (*                                                                  *)
  (*             NOW ONE NEEDS TO CHECK FOR SINGLE OPRANDS            *)
  (*           $ AND ¢ WHICH NEED ONLY OPRAND1                        *)
  (*                                                                  *)
  (*                                                                   *)
   IF(LRMATRIX(.L,K.).OPCODE) IN (. '$','¢' .) THEN
     BEGIN(*SINGLE OPRAND*)
     IF((LRMATRIX(.L,K.).ADDRESS1=I)AND (LRMATRIX(.L,K.).ADDOPR1=J))THEN
       BEGIN(*1*)
          MATRIX((LRMATRIX(.I,J.).COUNT),1);
       END;(*1*)
     END(*SINGLE OPRAND*)
  (*                                                                  *)
  (*             ELSE IF DOUBLE OPRAND                                *)
  (*                                                                  *)
   ELSE
    BEGIN(*DOUBLE OPRAND*)
  IF((LRMATRIX(.L,K.).ADDRESS1=I)AND (LRMATRIX(.L,K.).ADDOPR1=J))THEN
       BEGIN(*11*)
        MATRIX((LRMATRIX(.I,J.).COUNT),1);
```

```
515 |  IF((LRMATRIX(.L,K.).ADDRESS2=I)AND (LRMATRIX(.L,K.).ADDOPR2=J))THEN
516 |        BEGIN(*122*)
517 |         MATRIX((LRMATRIX(.I,J.).COUNT),2);
518 |        END(*122*)
519 |     END(*DOUBLE OPRAND*)
520 |        END(*14*)
521 |       END(*13*)
522 |      END(*12*)
523 |     END;(*11*)
524 |  (*                                                                  *)
525 |  (*                                                                  *)
526 |  (*                                                                  *)
527 |  (*         NOW EACH LEVEL IC HAS A COUNT OF HOW MANY                *)
528 |  (*  OTHER LOWER LEVEL IC NEED ITS OUTPUT. IT IS ASSUMED  THAT       *)
529 |  (*  THE NUMBER OF CELL BLOCKS AND ICS ARE ADEQUATE FOR THIS         *)
530 |  (*  PLACEMENT.                                                      *)
531 |  (*                                                                  *)
532 |  (*                                                                  *)
533 |  (*                                                                  *)
534 |  (*                                                                  *)
535 |  (*                                                                  *)
536 |  (*                                                                  *)
537 |  (*                                                                  *)
538 |  (*                                                                  *)
539 |  (*                                                                  *)
540 |  (*  NOW FORMING THE INSTRUCTION PACKET BY CHECKING EACH             *)
541 |  (*  LEVEL. THE PRIORITY STARTS FROM LEVEL 1                         *)
542 |  (*                                                                  *)
543 |  (*                                                                  *)
544 |          CBE:=1;ICE:=1;
545 |          FOR I:=1 TO LEVELINDEX DO
546 |            BEGIN(*A*)
547 |              FOR J:=1 TO COLMN(.I.) DO
548 |                BEGIN(*B*)
549 |  (*                                                                  *)
550 |  (*                CHECKING FOR LAST INSTRUCTION                     *)
551 |                IF ((I=LEVELINDEX)AND(J=COLMN(.I.)))THEN
552 |                  BEGIN (*LAST*)
553 |  CHANGEADDRESS(LRMATRIX,I,J,CBE,ICE,MAXMULT,UPCOUNT1,UPCOUNT2,
554 |                INDX1,INDX2);
555 |                     OPRD(LRMATRIX,I,J,CBE,ICE);
556 |                     CELLBLOCK(.CBE,ICE.).OPCODE:=LRMATRIX(.I,J.).OPCODE;
557 |                     CELLBLOCK(.CBE,ICE.).RESULT:=' ';
558 |                     CELLBLOCK(.CBE,ICE.).STAR:='A';
559 |  (*                                                                  *)
560 |  (*               NOW THIS RESULT SHOULD GO INTO FIN OPCODE           *)
561 |                   STR1:=CBE;
562 |                   STR2:=ICE;
563 |                   INCREMENT(CBE,ICE);
564 |                   CELLBLOCK(.STR1,STR2.).ADDRSLT1.CB:=CBE;
565 |                   CELLBLOCK(.STR1,STR2.).ADDRSLT1.IC:=ICE;
566 |                   CELLBLOCK(.STR1,STR2.).ADDRSLT1.OPR:=1;
567 |                   CELLBLOCK(.STR1,STR2.).ADDRSLT2.CB:=0;
568 |                   CELLBLOCK(.STR1,STR2.).ADDRSLT2.IC:=0;
569 |                   CELLBLOCK(.STR1,STR2.).ADDRSLT2.OPR:=0;
570 |  (*                                                                  *)
571 |  (*                NOW THE FIN OPCODE                                *)
572 |  (*                                                                  *)
573 |                   CELLBLOCK(.CBE,ICE.).OPCODE:='F';
574 |                   CELLBLOCK(.CBE,ICE.).STAR:='A';
575 |                   CELLBLOCK(.CBE,ICE.).OPRAND1:=' ';
576 |                   CELLBLOCK(.CBE,ICE.).OPRAND2:=' ';
577 |                   CELLBLOCK(.CBE,ICE.).RESULT:=' ';
```

```
580 |                       CELLBLOCK(.CBE,ICE.).ADDRSLT1.OPR:=0;
581 |                       CELLBLOCK(.CBE,ICE.).ADDRSLT2.CB:=0;
582 |                       CELLBLOCK(.CBE,ICE.).ADDRSLT2.IC:=0;
583 |                       CELLBLOCK(.CBE,ICE.).ADDRSLT2.OPR:=0
584 |                     END (*LAST*)
585 |                   ELSE(*NOT LAST*)
586 |                     BEGIN(*NOT LAST*)
587 |   (*                                                                  *)
588 |   (*                                                                  *)
589 |   (*                                                                  *)
590 |                     IF(LRMATRIX(.I,J.).COUNT=1)THEN
591 |                      BEGIN
592 |                      IF ((I=1)AND(J=1))THEN
593 |                      BEGIN
594 |   WRITELN('DEBUG');
595 |   WRITELN(LRMATRIX(.I,J.).ADDCOUNT(.1.).LEVELIND);
596 |   WRITELN(LRMATRIX(.I,J.).ADDCOUNT(.2.).LEVELIND);
597 |   WRITELN(LRMATRIX(.I,J.).ADDCOUNT(.1.).COLMNIND);
598 |   WRITELN(LRMATRIX(.I,J.).ADDCOUNT(.2.).COLMNIND);
599 |   WRITELN('DEBUG')
600 |   END;
601 |   CHANGEADDRESS(LRMATRIX,I,J,CBE,ICE,MAXMULT,UPCOUNT1,UPCOUNT2,
602 |                 'INDX1,INDX2);
603 |                     OPRD(LRMATRIX,I,J,CBE,ICE);
604 |                     CELLBLOCK(.CBE,ICE.).OPCODE:=LRMATRIX(.I,J.).OPCODE;
605 |                     CELLBLOCK(.CBE,ICE.).RESULT:=' ';
606 |                     CELLBLOCK(.CBE,ICE.).STAR:='A';
607 |                     TEMP:=LRMATRIX(.I,J.).COUNT;
608 |                     CELLBLOCK(.CBE,ICE.).ADDRSLT1.CB:=
609 |                     LRMATRIX(.I,J.).ADDCOUNT(.TEMP.).LEVELIND;
610 |                     CELLBLOCK(.CBE,ICE.).ADDRSLT1.IC:=
611 |                     LRMATRIX(.I,J.).ADDCOUNT(.TEMP.).COLMNIND;
612 |                     CELLBLOCK(.CBE,ICE.).ADDRSLT1.OPR:=
613 |                     LRMATRIX(.I,J.).ADDCOUNT(.TEMP.).OPRIND;
614 |                     CELLBLOCK(.CBE,ICE.).ADDRSLT2.CB:=0;
615 |                     CELLBLOCK(.CBE,ICE.).ADDRSLT2.IC:=0;
616 |                     CELLBLOCK(.CBE,ICE.).ADDRSLT2.OPR:=0;
617 |                     INCREMENT(CBE,ICE)
618 |                    END;
619 |   (*                                                                *)
620 |   (*                                                                  *)
621 |   (*                                                                  *)
622 |   (*            JUST ONE ADDRESS INSTRUCTION                          *)
623 |   (*                                                                  *)
624 |
625 |   (*                                                                  *)
626 |   (*                                                                  *)
627 |   (*                                                                  *)
628 |   (*                                                                  *)
629 |   (*              TWO ADDRESS INSTRUCTION                             *)
630 |   (*                                                                  *)
631 |   (*                                                                  *)
632 |                     IF(LRMATRIX(.I,J.).COUNT=2)THEN
633 |                      BEGIN
634 |   WRITELN('DEBUG');
635 |   WRITELN(LRMATRIX(.I,J.).ADDCOUNT(.1.).LEVELIND);
636 |   WRITELN(LRMATRIX(.I,J.).ADDCOUNT(.2.).LEVELIND);
637 |   WRITELN(LRMATRIX(.I,J.).ADDCOUNT(.1.).COLMNIND);
638 |   WRITELN(LRMATRIX(.I,J.).ADDCOUNT(.2.).COLMNIND);
639 |   WRITELN('DEBUG');
640 |   CHANGEADDRESS(LRMATRIX,I,J,CBE,ICE,MAXMULT,UPCOUNT1,UPCOUNT2,
641 |                 INDX1,INDX2);
642 |                     OPRD(LRMATRIX,I,J,CBE,ICE);
```

```
    645 |                    CELLBLOCK(.CBE,ICE.).STAR:='B';
    646 |                    TEMP:=(LRMATRIX(.I,J.).COUNT)-1;
    647 |                    CELLBLOCK(.CBE,ICE.).ADDRSLT1.CB:=
    648 |                    LRMATRIX(.I,J.).ADDCOUNT(.TEMP.).LEVELIND;
    649 |                    CELLBLOCK(.CBE,ICE.).ADDRSLT1.IC:=
    650 |                    LRMATRIX(.I,J.).ADDCOUNT(.TEMP.).COLMNIND;
    651 |                    CELLBLOCK(.CBE,ICE.).ADDRSLT1.OPR:=
    652 |                    LRMATRIX(.I,J.).ADDCOUNT(.TEMP.).OPRIND;
    653 |  CELLBLOCK(.CBE,ICE.).ADDRSLT2.CB:=LRMATRIX(.I,J.).ADDCOUNT(.(TEMP+1).).LEVELIND;
    654 |  CELLBLOCK(.CBE,ICE.).ADDRSLT2.IC:=LRMATRIX(.I,J.).ADDCOUNT(.(TEMP+1).).COLMNIND;
    655 |  CELLBLOCK(.CBE,ICE.).ADDRSLT2.OPR:=LRMATRIX(.I,J.).ADDCOUNT(.(TEMP+1).).OPRIND;
    656 |   INCREMENT(CBE,ICE)
    657 |                   END;
    658 |                END;(*NOT LAST*)
    659 |                 END;(*B*)
    660 |                END;(*A*)
    661 |  (*                                                                  *)
    662 |  (*                                                                  *)
    663 |  (*                                                                  *)
    664 |    WRITELN
    665 |  ('FORMED  M E M O R Y  I N S T R U C T I O N S  BY STAGE2':60);
    666 |  WRITELN
    667 |  ('REMEMBER:THE FINISH OPCODE IS REPRESENTED BY    F ' :60);
    668 |  WRITELN('REMEMBER:THE BLANK IS REPRESENTED BY     UNLIKE  99':60);
    669 |    FOR I:=1 TO MAXCELBLK DO
    670 |     BEGIN
    671 |       FOR J:=1 TO MAXIC DO
    672 |         RITEM(CELLBLOCK,I,J);
    673 |     END;
    674 |  END.
Execution begins...
DEBUG
           2
           0
           1
           0
DEBUG
DEBUG
           2
           2
           1
           2
DEBUG
     FORMED  M E M O R Y  I N S T R U C T I O N S  BY STAGE2
          REMEMBER:THE FINISH OPCODE IS REPRESENTED BY    F
         REMEMBER:THE BLANK IS REPRESENTED BY     UNLIKE  99
   ************************************************************
                            CELLBLOCK   1 |INST.NO 1
                       CELLBLOCK.OPCODE    +
                      CELLBLOCK.OPRAND1  4
                      CELLBLOCK.OPRAND2  1
                       CELLBLOCK.RESULT
                         CELLBLOCK.STAR   A
                    CELLBLOCK.ADDRSLT1.CB  2
                    CELLBLOCK.ADDRSLT1.IC  2
                   CELLBLOCK.ADDRSLT1.OPR  1
                    CELLBLOCK.ADDRSLT2.CB  0
                    CELLBLOCK.ADDRSLT2.IC  0
                   CELLBLOCK.ADDRSLT2.OPR  0
   ************************************************************
                            CELLBLOCK   1 |INST.NO 2
                       CELLBLOCK.OPCODE    -
                      CELLBLOCK.OPRAND1  3
```

```
                    CELLBLOCK.RESULT
                      CELLBLOCK.STAR    A
                  CELLBLOCK.ADDRSLT1.CB  1
                  CELLBLOCK.ADDRSLT1.IC  3
                 CELLBLOCK.ADDRSLT1.OPR  2
                  CELLBLOCK.ADDRSLT2.CB  0
                  CELLBLOCK.ADDRSLT2.IC  0
                 CELLBLOCK.ADDRSLT2.OPR  0
*********************************************************
                        CELLBLOCK    1 |INST.NO 3
                   CELLBLOCK.OPCODE     +
                  CELLBLOCK.OPRAND1
                  CELLBLOCK.OPRAND2
                   CELLBLOCK.RESULT
                     CELLBLOCK.STAR    A
                  CELLBLOCK.ADDRSLT1.CB  2
                  CELLBLOCK.ADDRSLT1.IC  3
                 CELLBLOCK.ADDRSLT1.OPR  2
                  CELLBLOCK.ADDRSLT2.CB  0
                  CELLBLOCK.ADDRSLT2.IC  0
                 CELLBLOCK.ADDRSLT2.OPR  0
*********************************************************
                        CELLBLOCK    1 |INST.NO 4
                   CELLBLOCK.OPCODE     F
                  CELLBLOCK.OPRAND1
                  CELLBLOCK.OPRAND2
                   CELLBLOCK.RESULT
                     CELLBLOCK.STAR    A
                  CELLBLOCK.ADDRSLT1.CB  0
                  CELLBLOCK.ADDRSLT1.IC  0
                 CELLBLOCK.ADDRSLT1.OPR  0
                  CELLBLOCK.ADDRSLT2.CB  0
                  CELLBLOCK.ADDRSLT2.IC  0
                 CELLBLOCK.ADDRSLT2.OPR  0
*********************************************************
                        CELLBLOCK    2 |INST.NO 1
                   CELLBLOCK.OPCODE     *
                  CELLBLOCK.OPRAND1  2
                  CELLBLOCK.OPRAND2  1
                   CELLBLOCK.RESULT
                     CELLBLOCK.STAR    B
                  CELLBLOCK.ADDRSLT1.CB  2
                  CELLBLOCK.ADDRSLT1.IC  2
                 CELLBLOCK.ADDRSLT1.OPR  2
                  CELLBLOCK.ADDRSLT2.CB  1
                  CELLBLOCK.ADDRSLT2.IC  3
                 CELLBLOCK.ADDRSLT2.OPR  1
*********************************************************
                        CELLBLOCK    2 |INST.NO 2
                   CELLBLOCK.OPCODE     -
                  CELLBLOCK.OPRAND1
                  CELLBLOCK.OPRAND2
                   CELLBLOCK.RESULT
                     CELLBLOCK.STAR    A
                  CELLBLOCK.ADDRSLT1.CB  2
                  CELLBLOCK.ADDRSLT1.IC  3
                 CELLBLOCK.ADDRSLT1.OPR  1
                  CELLBLOCK.ADDRSLT2.CB  0
                  CELLBLOCK.ADDRSLT2.IC  0
                 CELLBLOCK.ADDRSLT2.OPR  0
*********************************************************
                        CELLBLOCK    2 |INST.NO 3
                   CELLBLOCK.OPCODE     /
                  CELLBLOCK.OPRAND1
```

```
                    CELLBLOCK.RESULT
                      CELLBLOCK.STAR    A
                  CELLBLOCK.ADDRSLT1.CB   1
                  CELLBLOCK.ADDRSLT1.IC   4
                 CELLBLOCK.ADDRSLT1.OPR   1
                  CELLBLOCK.ADDRSLT2.CB   0
                  CELLBLOCK.ADDRSLT2.IC   0
                 CELLBLOCK.ADDRSLT2.OPR   0
  ************************************************************
                           CELLBLOCK    2 |INST.NO 4
                    CELLBLOCK.OPCODE
                   CELLBLOCK.OPRAND1
                   CELLBLOCK.OPRAND2
                    CELLBLOCK.RESULT
                      CELLBLOCK.STAR    D
                  CELLBLOCK.ADDRSLT1.CB   0
                  CELLBLOCK.ADDRSLT1.IC   0
                 CELLBLOCK.ADDRSLT1.OPR   0
                  CELLBLOCK.ADDRSLT2.CB   0
                  CELLBLOCK.ADDRSLT2.IC   0
                 CELLBLOCK.ADDRSLT2.OPR   0

...execution ends
File 'GENIC2': 674 lines; no diagnostics
35152 bytes of object code generated
1460 statements executed
59944 bytes of memory requested during compilation
5792 bytes returned before execution
10672 bytes requested during execution
```

APPENDIX F

OUTPUT DESCRIPTION OF THE UPDATE UNIT

DESCRIPTION OF OUTPUT GENERATION:

For simplicity th following notation is followed:

ADDR I.ADDR 1.IC:=V, ADDR I. ADDR 1.IC:=V'

ADDR I.ADDR 2.IC:=Q, -DO-

ADDR II.ADDR 1.IC:=S -DO-

ADDR II. ADDR 2.IC:=T -DO-

The case statement represents the first stage demux while the statements inside the BEGIN and END represent the second stage demux.

DESCRIPTION OF OUTPUT GENERATION FOR UPDATE UNIT

CASE

AA:

```
IF V'AND S' TRUE THEN
 BEGIN
   S2=0;LOAD1=0;(*SINCE BOTH ADDRESSES ARE NON VALID*)
   S1=ON;LOADADD1=0000;
   S4=0;
   LOAD IC DEMUX I=0000
```

```
   LOAD IC DEMUXII=0000
 END;

IF V' AND S TRUE THEN
     BEGIN
          S2=0;LOAD1=1;
          S1=1;LOAD ADD1=ADDRII.ADDR1.IC;
          S4=0;
          LOAD IC DEMUX I:=0000;
          LOAD IC DEMUX II:=ADDRI.ADDR1.IC;
     END;

IF V AND S' TRUE THEN

     BEGIN
          S2=0;LOAD1=1;
          S1=1;LOAD ADD1=ADDRI.ADDR1.IC;
          S4=0;
          LOAD IC DEMUX I:=ADDRI.ADDR1.IC;
          LOAD IC DEXUM II:=0000;
     END;

IF  V AND S TRUE THEN
     BEGIN
          S2=1;LOAD21=LOAD22=1;
          S1=S4=0;
          LOAD ADD 21=ADDRII.ADDR1.IC;
          LOAD ADD 22=ADDRI. ADDR1.IC;
          LOAD IC DEMUX I:=ADDRI.ADDR1.IC;
```

```
          LOAD IC DEMUX II=ADDRII.ADDR1.IC
     END;
AB:
IF (V' AND S' AND T ') TRUE THEN
     BEGIN
          LOAD 1=0;
          LOAD ADD1=0000;
          S1=1;S2=0;
          LOAD IC DEMUXI=0000;
          LOAD IC DEMUX II=0000;
          S4=0;
     END;
IF (V' AND S ' AND T)TRUE THEN
     BEGIN
          LOAD1=1;
          LOAD ADD1 = ADDR II.ADDR 2.IC;
          S1=1;S2=0;
          LOAD IC DEMUX I=0000;
          LOAD IC DEMUX II= ADDR II. ADDR 2.IC;
          S4=0;
     END;
IF (V' AND S AND T') TRUE THEN
     BEGIN
          LOAD 1=1;
```

```
        LOAD ADD 1= ADDR II. ADDR1.IC;
        S1=1;S2=0;S4=0;
        LOAD IC DEMUX I:=0000;
        LOAD IC DEMUX II=ADDR II. ADDR 1. IC;
    END;

IF (V' AND S AND T )TRUE THEN

    BEGIN
        LOAD1=0;
        S1=0;S2=1;S4=1;(SAME RESULT NEED TO BE UPDATED*)
        LOAD ADD 21=ADDR II. ADDR 1. IC;
        LOAD ADD 22= ADDR II. ADDR 2. IC
        LOAD 21= LOAD 22= 1;
        LOAD IC EEMUX I:=ADDR II. ADDR 2. IC;
        LOAD IC DEMUX II:= ADDR II. ADDR 1. IC;
    END;

IF (V AND S' AND T')TRUE THEN

    BEGIN
        LOAD 1= 1;S1=1;S2=S4=0;
        LOAD ADD1:= ADDR I. ADDR 1. IC;
        LOAD IC DEMUX I:= ADDR I. ADDR 1. IC;
        LOAD IC DEMUX II:= 0000;
    END;

IF (V AND S' AND T) TRUE THEN

    BEGIN
```

```
          LOAD 1=0;S1=0;S2=1;S4=0;
          LOAD 21=LOAD 22=1;
          LOAD ADD 21 = ADDR II. ADDR 2. IC;
          LOAD ADD 22:= ADDR I. ADDR 1. IC;
          LOAD IC DEMUX I:= ADDR I. ADDR. IC
          LOAD IC DEMUX II:= ADDR II. ADDR 2. IC;
     END;

IF (V AND S AND T')TRUE THEN

     BEGIN
          LOAD 1 =0;S1=0;S2=1;S4=0;
          LOAD 21=LOAD 22=1;
          LOAD ADD 21= ADDR II. ADDR 1. IC;
          LOAD ADD 22:=ADDR I. ADDR 1. IC;
          LOAD IC DEMUX I:=ADDR I. ADDR 1. IC;
          LOAD IC DEMUX II:=ADDR II. ADDR 1. IC;
     END;

IF ( V AND S AND T) TRUE THEN
     BEGIN
          UPDATE QUEUE OVF:=1;
     END;(*QUEUE OVERFLOW*)


AC:


IF (V' AND  S')TRUE THEN
```

```
      BEGIN
            S2=0;S1=1;LOAD 1=1;LOAD ADD1 :=0000;
            S4=0;
            LOAD IC DEMUX I:=0000;
            LOAD IC DEMUX II:=0000;
      END;

IF ( V' AND S )TRUE THEN
      BEGIN
            S2=0;S1=1;LOAD 1=1;LOAD ADD 11=ADDR II. ADDR 1. IC;
            S4=0;
            LOAD IC DEMUX I:=0000;
            LOAD IC DEMUXII:= ADDR II. ADDR 1. IC;
      END;


IF ( V AND S') TRUE THEN
      BEGIN

            S2=0;S1=1;S4=0;;LOAD 1=1;LOAD ADD1 :=ADDR I. ADDR 1. IC;
            LOAD IC DEMUX I:= ADDR I. ADDR 1. IC;
            LOAD IC DEMUX II:= 0000;
      END;

IF (V AND S ) TRUE THEN
      BEGIN

            S2=1;S1=0;LOAD 21 = LOAD 22:=1;
            S4=0;
            LOAD ADD 21:=ADDR II. ADDR 1. IC;
```

```
        LOAD ADD 22:= ADDR I. ADDR 1. IC;
        LOAD IC DEMUX I:= ADDR I. ADDR 1. IC;
        LOAD IC DEMUX II := ADDR II. ADDR 1. IC;
    END;


AD:

IF (V ' AND  T' ) TRUE THEN
    BEGIN

        S2=0;S1-1;LOAD1=0;LOAD ADD 1:=0000;S4=0;
        LOAD IC DEMUX I:=0000;
        LOAD IC DEMUX ii:=0000;
    END;

IF ( V' AND T ) TRUE THEN

    BEGIN

        S2=0;S1=1;LOAD 1=1;LOAD ADD1:=ADDR II. ADDR 2. IC;
        S4=0;
        LOAD IC DEMUX I:=0000;
        LOAD IC DEMUX ii:= ADDR II. ADDR 2. IC;
    END;

IF (V AND T') TRUE THEN

    BEGIN

        S2=0;S1=1;LOAD 1=1;LOAD ADD 1:=ADDR I. ADDR 1. IC;
        S4=0;
```

```
        LOAD IC DEMUX I:=ADDR I. ADDR 1. IC;
        LOAD IC DEMUX II:=0000;

    END;


IF ( V AND T ) TRUE THEN

    BEGIN

        S2=1;S1=0; LOAD 21=LOAD 22=1;
        S4=0;
        LOAD ADD 21=ADDR II. ADDR 2. IC;
        LOAD ADD 22:= ADDR I. ADDR 1. IC;
        LOAD IC DEMUX I:= ADDR I. ADDR 1 . IC;
        LOAD IC DEMUX II:= ADDR II. ADDR 2. IC ;
    END;



BA:

IF ( V' AND Q' AND S') TRUE THEN

    BEGIN

        LOAD1=0;
        S1=1;S2=0;S4=0;LOAD ADD 1:=0000;
        LOAD IC DEMUX I:=0000;
        LOAD IC DEMUX II:=0000;
    END;
```

```
IF (V ' AND Q' AND S) TRUE THEN

     BEGIN

          S1=1;S2=0;S4=0;LOAD 1:=1;

          LOAD ADD1:=ADDR II. ADDR 1. IC;

          LOAD IC DEMUX I:= ADDR I. ADDR2. IC;

          LOAD IC DEMUX II:=0000;

     END;

IF (V' AND Q AND S') TRUE THEN

     BEGIN

          S1=1;S2=0;LOAD 11=1;LOAD ADD1:=ADDR I. ADDR 2.IC;

          S4=0;

          LOAD IC DEMUX I:=ADDRI. ADDR 2. IC;

          LOAD IC DEMUX II:=0000;

     END;

IF (V' AND Q AND S) TRUE THEN

     BEGIN

          LOAD 1:=0;S1=0;LOAD 21=LOAD 22:=1;S2=1;

          LOAD ADD 21:=ADDR II. ADDR 1. IC;

          LOAD ADD 22:=ADDR I. ADDR2. IC;

          S4:=0;LOAD IC DEMUX I:=ADDR I. ADDR 2. IC;

          LOAD IC DEMUX II:= ADDR II. ADDR 1. IC;

     END;

IF (V AND Q' AND S') TRUE THEN
```

```
     BEGIN

          LOAD 1:= 1;S1:=1;S2=0;S4:=0;

          LOAD ADD1:=ADDRI. ADDR 1. IC;

          LOAD IC DEMUX I:= ADDR I. ADDR 1. IC;

          LOAD IC DEMUM II:=0000;

     END;

IF (V AND Q' AND S ) TRUE THEN

     BEGIN

          S1:=0;LOAD 1:= 0; S2:=1;S4:=0;

          LOAD 21= LOAD 22:=1;

          LOAD ADD 21:= ADDR II. ADDR 1. IC;

          LOAD ADD 22:= ADDR I. ADDR 1. IC;

          LOAD IC DEMUX I:= ADDR I. ADDR 1. IC;

          LOAD IC DEMUX II:= ADDR II. ADDR 1. IC;

     END;

IF ( V AND Q AND S') TRUE THEN

     BEGIN

          S1=0;LOAD 1:=0;S2=1; LOAD 21 =LOAD 221= 1;

          LOAD ADD21:=ADDRI.ADDR2.IC;

          LOAD ADD22:=ADDRI.ADDR1.IC;

          S4:=1(*SAME RESULT *)

          LOAD IC DEMUX I:= ADDR I. ADDR 1. IC;

          LOAD IC DEMUX II:= ADDDR I. ADDR2. IC;
```

```
      END;

IF ( V AND Q AND S ) TRUE THEN

       BEGIN

                UPDATE QUEUE OVF:=1;

                (*QUEUE OVERFLOW*)

       END;



BB:

IF ( V' AND Q' AND S' AND T') TRUE THEN

      BEGIN

           S1=1;S2:=0;S4:=0;

           LOAD 1:=0;LOAD ADD1:=0000;

           LOAD IC DEMUX I:=0000;

           LOAD IC DEMUX II:=0000

      END;

IF ( V' AND Q' AND S' AND T) TRUE THEN

      BEGIN

           S1=0;S2=1;LOAD1=1;

           S4=0;

           LOAD ADD1:= ADDR II. ADDR2.IC;
```

```
          LOAD IC DEMUX I:=0000;
          LOAD IC DEMUX II:= ADDR II. ADDR 2. IC;

     END;

IF (V' AND Q' AND S AND T')TRUE THEN

     BEGIN
          S1=1;S2=0; LOAD 1:=1;
          S4:=0;LOAD ADD 1:= ADDR II. ADR 1. IC;
          LOAD 1:= 1;
          LOAD IC DEMUX I:=. 0000;
          LOAD IC DEMUM II:= ADDR II. ADDR 1. IC;

     END;

IF ( V' AND Q' AND S AND T ) TRUE THEN

     BEGIN

          S1=0;S2=1;LOAD 21=LOAD22:= 1; S4:= 1;(*SAME RESULT*)
          LOAD ADD1:= ADDR II. ADDR 1. IC;
          LOAD ADD2:= ADDR II. ADDR 2. IC;
          LOAD IC DEMUX I:= ADDR II. ADDR1 .IC;
          LOAD IC DEMUM II:= ADDR II. ADDR2. IC;
     END;

IF (V' AND Q AND S' AND T')TRUE THEN

     BEGIN

          S1=1;S2=0;LOAD 1:= 1;
```

```
        S4:=0;LOAD ADD 1:= ADDR I. ADDR 2. IC;
        LOAD IC DEMUX I:= ADDR I. ADDR2.IC;
        LOAD IC DEMUX II:=0000;
    END;

IF( V' AND Q AND S' AND T )TRUE THEN
    BEGIN
        S2=1;S1=0;LOAD21=LOAD 22=1; S4=0;
        LOAD ADD 21:= ADDR II. ADDR 2. IC;
        LOAD ADD 22:= ADDR I. ADDR2. IC;
        LOAD IC DEMUX I= ADDR I. ADDR2. IC;
        LOAD IC DEMUX I:= ADDR II. ADDR 2. IC;
    END;

IF (V' AND Q AND S AND T' ) TRUE THEN
    BEGIN
        LOAD 21:=LOAD 22= 1; S2=1; S1=0; S4=0;
        LOAD ADD 21:= ADDR I. ADDR2. IC;
        LOAD ADD 22;= ADDR II. ADDR 1. IC;
        LOAD IC DEMUX I:= ADDR I. ADDR 2. IC;
         LOAD IC DEMUX II:= ADDR II . ADDR 2. IC;
    END;

IF (V' AND Q AND S AND T ) TRUE THEN
    BEGIN
        UPDATE QUEUE OVF:=1;
```

```
      END;

IF ( V AND Q' AND S' AND T') TRUE THEN

      BEGIN

            S1= 1;S2=0;S4=0;

            LOAD 1:=1;

            LOAD ADD1:= ADDRI. ADDR 1. IC;

            LOAD IC DEMUX I:= ADDR I. ADDR 1. IC;

            LOAD IC DEMUX II:= 0000;

      END;

IF (V AND Q' AND S' AND T ) TRUE THEN

      BEGIN

            S1=0;S2=1;;LOAD 21=LOAD 22=1;

            LOAD ADD21= ADDR I. ADDR 1. IC;

            LOAD ADD 22:= ADDR II. ADDR 2. IC;

            S4=0;

            LOAD IC DEMUX I:= ADDR I. ADDR 1. IC;

            LOAD IC DEMUX II:= ADDR II. ADDR 2. IC;

      END;

IF (V AND Q' AND S AND T' ) TRUE THEN

      BEGIN

            S1=0; S2 =1;

            LOAD 21=LOAD22:=1;

            S4=0;
```

```
        LOAD ADD21:=ADD II ADD 1 IC;LOAD ADD22:=ADDR I.
                        ADDR 1.IC;
        LOAD IC DEMUX I:= ADDR I. ADDR 1. IC;
        LOAD IC DEMUX II:= ADDR II. ADDR1 .IC;

    END;

IF( V AND Q' AND S AND T ) TRUE THEN
    BEGIN
        UPDATE QUEUE OVF :=1;
    END;

IF(V AND Q AND S' AND T' ) TRUE THEN

    BEGIN

        S2=1;S1=0;S4=1;
        LOAD 21=LOAD 22=1;.
        LOAD ADD 21:= ADDR I. ADDR1. IC;
        LOAD ADD 22:= ADDR I.ADDR 2. IC;
        LOAD IC DEMUX I:= ADDR I. ADDR 1. IC;
        LOAD IC DEMUX II:= ADDR I. ADDR 2.IC;

    END;

IF ( V AND Q AND S' AND T ) TRUE THEN
    BEGIN
        UPDATE QUEUE OVF:=1;
    END;
```

```
IF ( V AND Q AND S AND T' ) TRUE THEN
      BEGIN
          UPDATE QUEUE OVF:=1;
     END;

IF (V AND Q AND S AND T ) TRUE THEN
     BEGIN
          UPDATE QUEUE OVF:=1;
     END;


BC:

IF(V'AND Q' AND S')TRUE THEN

     BEGIN

          S1=1;S2=0; S4=0;
          LOADADD1:=0000;
          LOAD IC DEMUX I:=0000;LOAD IC DEMUX II:=0000;
     END;

IF ( V' AND Q' AND S) TRUE THEN

      BEGIN
           S1=1;S2=0;S4=0;LOAD1:=TRUE
          LOAD ADD1 := ADDR II. ADDR1. IC;
          LOAD IC DEMUX I:=0000;

          LOAD IC DEMUX II:=ADDRII.ADDR1.IC;
     END;
```

```
IF ( V' AND Q AND S' ) TRUE THEN

     BEGIN

          S1=1;S2=0;LOAD 1:= ADDR I. ADDR2. IC;

          LOAD ADD 1:=ADDR I. ADDR 2. IC;

          LOAD IC DEMUX I:= ADDR I. ADDR2. IC;

          LOAD IC DEMUX II:= 0000;

     END;

IF ( V' AND Q AND S ) TRUE THEN

     BEGIN

          S1=0; S2=1; LOAD21=LOAD22=1;

          LOAD ADD 21:= ADDR II. ADDR1. IC;

          LOAD ADD 22:= ADDR I. ADDR2.IC;

          S4=0;

          LOAD IC DEMUX I:=ADDR I. ADDR2. IC;

          LOAD IC DEMUX II:= ADDR II. ADDR 1. IC;

     END;

IF( V AND Q' AND S') TRUE THEN

     BEGIN

          S1=1;S2=0;LOAD1=1;LOAD ADD1:= ADDR I.ADDR1.IC;

          LOAD IC DEMUX I:= ADDR I. ADR 1.IC;

          LOAD IC DEMUX II:=0000;

          S4:=0;

     END;
```

```
IF( V AND Q' AND S) TRUE THEN

     BEGIN

          S2=1;S1=0;S4=0;

          LOAD 21=LOAD22=1;

          LOAD  ADD 21:= ADDR II. ADDR 1. IC

          LOAD ADD 22:= ADDR I. ADDR1. IC;

          LOAD IC DEMUX I:= ADDR I. ADDR1. IC;

          LOAD IC DEMUX II:=ADDR II. ADDR1.IC;

     END;

IF ( V AND Q AND S') TRUE THEN

     BEGIN

          S2=1;S1=0;S4=1;(*SAME RESULT*)

          LOAD 21=LOAD 22:=1;

          LOAD ADD21:= ADDRI . ADDR 1.IC;

          LOAD ADD 22:= ADDR I. ADDR 2. IC;

          LOAD IC DEMUX I:=ADDR I. ADDR 1.IC;

          LOAD IC DEMUX II:= ADDRI. ADDR 2.IC;

     END;

IF(V AND Q AND S )TRUE THEN

     BEGIN

          UPDATE QUEUE OVF:=1;

     END;
```

```
BD;

IF (V' AND Q' AND T' ) TRUE THEN

     BEGIN
          S1=1;S2=0;S4=0;
          LOAD 1:=0;
          LOAD ADD 1:=0000;
          LOAD IC DEMUX I:=0000;
          LOAD IC DEMUX II:=0000;
     END;

IF ( V' AND Q' AND T ) TRUE THEN

     BEGIN
          S1=1;S2=0;S4=0;
          LOAD 1:=1;
          LOAD ADD 1:=ADDR II. ADDR 2. IC;
          LOAD IC DEMUXI:=0000;
          LOAD IC DEMUX II:=ADDR II. ADDR2. IC;
     END;

IF ( V'AND Q AND T')TRUE THEN

     BEGIN
          S1=1;S2=0;S4=0;
          LOAD 1:=1;LOAD ADD1:= ADDRI.ADDR2.IC;
          LOAD IC DEMUM I:= ADDRI.ADDR2.IC;
```

```
        LOAD IC DEMUX II:=0000;
    END;

IF(V' AND Q AND T) TRUE THEN

    BEGIN
        S1=0;S2=1;S4=0;
        LOAD21=LOAD22:=1;
        LOAD ADD21:=ADDRII. ADDR2.IC;
        LOAD ADD 22:=ADDRI. ADDR2.IC;
        LOAD IC DEMUXI:=ADDRI. ADDR2.IC;
        LOAD IC DEMUX II:=ADDR II. ADDR2.IC;
    END;

IF (V AND Q' AND T') TRUE THEN

    BEGIN
        S1=1;S2=0;S4=0;
        LOAD1:=1;
        LOAD ADD1:=ADDRI. ADDR1.IC;
        LOAD IC DEMUXI:=ADDRI. ADDR1.IC;
        LOAD IC DEMUX II:=0000;
    END;

IF (V AND Q'AND T)TRUE THEN
    BEGIN
        S1=0;S2=1;S4=0;
        LOAD 21:=LOAD 22:=1;
        LOAD ADD 21:=ADDRII. ADDR2. IC;
        LOAD ADD 22:= ADDR I. ADDR1. IC;
```

```
        LOAD IC DEMUX I:= ADDR I. ADDR 1. IC;
        LOAD IC DEMUX II:= ADDR II. ADDR 2. IC;
    END;

IF( V AND Q AND T') TRUE THEN

    BEGIN
        S1=0;S2=1;S4=1;
        LOAD 21:=LOAD 22:=1;
        LOAD ADD 21:= ADDRI. ADDR 1. IC;
        LOAD ADD 22:= ADDR I. ADDR 2. IC;
        LOAD IC DEMUX I:=ADDRI. ADDR1. IC;
        LOAD IC DEMUX II:=ADDR 1. ADDR2.IC;
    END;


IF ( V AND Q AND T ) TRUE THEN
    BEGIN
        UPDATE QUEUE OVF:=1;
    END;


CA:

IF ( V' AND S' )TRUE THEN
    BEGIN

        S1=1;S2=0;S4=0;
        LOAD 1:=0;
        LOAD ADD 1:=0000;
```

```
          LOAD IC DEMUX I:=0000;
          LOAD IC DEMUX II:=0000;

     END;

IF ( V'AND S) TRUE THEN

     BEGIN

          S1=1;S2=0;S4=0;
          LOAD 1:=1;
          LOAD ADD1:=ADDR II. ADDR1. IC;
          LOAD IC DEMUX I:=0000;
          LOAD IC DEMUX II:=ADDR II.ADDR 1. IC;

     END;

IF ( V AND S') TRUE THEN

          BEGIN

               S1=1;S2=0;S4=0;
               LOAD 1:=1;
               LOAD ADD1:=ADDR I. ADDR1. IC;
               LOAD IC DEMUXI:=ADDR I. ADDR 1. IC;
               LOAD IC DEMUX II:=0000;
          END;

IF (V AND S ) TRUE THEN

     BEGIN

          S1=0;S2=1;S4=0;
```

```
      BEGIN
          S1=1;S2=0;S4=0;
          LOAD1:=1;
          LOAD ADD1:=ADDRII. ADDR1. IC;
          LOAD IC DEMUXI:=0000;
          LOAD IC DEMUX II:= ADDR II. ADDR1. IC;
     END;
IF( V' AND S AND T ) TRUE THEN
     BEGIN
          S1=0;S2=1;S4=1;
          LOAD 21:=LOAD 22:=1;
          LOAD ADD21:=ADDR II. ADDR1. IC;
          LOAD ADD22:=ADDR II. ADDR2. IC;
          LOAD IC DEMUX I:=ADDR II. ADDR1.IC;
          LOAD IC DEMUX II:=ADDR II.ADDR2.IC;
     END;
IF( V AND S' AND T')TRUE THEN
     BEGIN
          S1=1;S2=0;S4=0;
          LOAD 1:=1;
          LOAD ADD 1:= ADDRI. ADDR1.IC;
          LOAD IC DEMUX I:=ADDRI .ADDR1 . IC;
          LOAD IC DEMUX II:=0000;
```

```
     END;

 IF(V AND S' AND T ) TRUE THEN

     BEGIN
          S1=0;S2=1;S4=0;
          LOAD 21:=LOAD 22:=1;
          LOAD ADD21:=ADDR II. ADDR2. IC;
          LOAD ADD22:=ADDR I. ADDR 1.IC;
          LOAD IC DEMUX I:=ADDR I. ADDR1. IC;
          LOAD IC DEMUX II:=ADDR II. ADDR2 .IC;
     END;

IF(V AND S AND T' ) TRUE THEN
     BEGIN

          S1=0;S2=1;S4=0;
          LOAD 21:=LOAD 22:=1;
          LOAD ADD 21:=ADDR II. ADDR1.IC;
          LOAD ADD 22:=ADDRI. ADDR1.IC;
          LOAD IC DEMUX I:=ADDRI. ADDR1. IC;
          LOAD IC DEMUX II:=ADDRII. ADDR1. IC;
     END;

IF (V AND S AND T) TRUE THEN
     BEGIN
          UPDATE QUEUE OVF:=1;
     END;

CC:
```

```
IF( V' AND S') TRUE THEN

     BEGIN

          S1=1;S2=0;S4=0;

          LOAD 1:=0;

          LOAD ADD1:=0000;

          LOAD IC DEMUX I:=0000;

          LOAD IC DEMUX II:=0000;

     END;

IF ( V' AND S ) TRUE THEN

     BEGIN

          S1=1;S2=0;S4=0;

          LOAD 1:=1;

          LOAD ADD1:=ADDRII..ADDR 1. IC;

          LOAD IC DEMUX I:=0000;

          LOAD IC DEMUX II:=ADDR II. ADDR1. IC;

     END;

IF( V AND S') TRUE THEN

     BEGIN

          S1=1;S2=0;S4=0;

          LOAD 1:=1;

          LOAD ADD1:=ADDRI. ADDR1. IC;

          LOAD IC DEMUX I:=ADDRI. ADDR1. IC;

          LOAD IC DEMUX II:=0000;
```

```
      END;

IF( V AND S ) TRUE THEN

      BEGIN

            S1=0;S2=1;S4=0;

            LOAD 21:=LOAD 22:= 1;

            LOAD ADD 21:=ADDR II. ADDR1. IC;

            LOAD ADD 22:=ADDR I.ADDR 1. IC;

            LOAD IC DEMUX I:= ADDR I. ADDR1. IC;

            LOAD IC DEMUX II:=ADDR II.ADDR1.IC;

      END;

CD:

IF (V' AND T' ) TRUE THEN

    BEGIN

            S1=1;S2=0;S4=0;

            LOAD 1:=0;

            LOAD ADD 1:=0000;

            LOAD IC DEMUX I:=0000;

            LOAD IC DEMUX II:=0000;

      END;

IF ( V' AND T ) TRUE THEN

            BEGIN

            S1=1;S2=0;S4=0;
```

```
        LOAD 1:=1;
        LOAD ADD1:=ADDRII.ADDR2. IC;
        LOAD IC DEMUX I:=0000;
        LOAD IC DEMUX II:=ADDRII . ADDR2. IC;

        END;

IF( V AND T') TRUE THEN

    BEGIN

        S1=1;S2=0;S4=0;
        LOAD 1:=1;
        LOAD ADD1:=ADDRI.ADDR1. IC;
        LOAD IC DEMUX I:=ADDRI. ADDR1. IC;
        LOAD IC DEMUX II:=0000;
    END;

IF( V AND T ) TRUE THEN

    BEGIN

        S1=0;S2=1;S4=0;
        LOAD 21:=LOAD 22:=1;
        LOAD ADD21:=ADDRII. ADDR2. IC;
        LOAD ADD 22:=ADDR I. ADDR1.IC;
        LOAD IC DEMUX I:=ADDRI. ADDR1. IC;
        LOAD IC DEMUXII:=ADDR II. ADDR2.IC;
    END;
```

```
DA:

IF (Q' AND S')TRUE THEN

     BEGIN
          S1=1;S2=0;S4=0;
          LOAD 1:=0;LOAD ADD 1:=0000;
          LOAD IC DEMUX I:=0000;
          LOAD IC DEMUX II:=0000;
     END;

IF( Q' AND S )TRUE THEN

     BEGIN
          S1=1;S2=0;S4=0;
          LOAD 1:=1;LOAD ADD1:=ADDRII. ADDR1. IC;
          LOAD IC DEMUX I:=0000;
          LOAD IC DEMUX II:=ADDRII.ADDR1.IC;
     END;

IF(Q AND S') TRUE THEN

     BEGIN
          S1=1;S2=0;S4=0;
          LOAD 1:=1;LOAD ADD1:=ADDRI. ADDR2. IC;
          LOAD IC DEMUX I:=ADDR I. ADDR2. IC;
          LOAD IC DEMUX II:=0000;

     END;

IF( Q AND S )TRUE THEN
```

```
     BEGIN
          S1=0;S2=1;S4:=0;
          LOAD 21:=LOAD 22:=1;
          LOAD ADD 21:=ADDRII. ADDRI.IC;
          LOAD ADD 22:=ADDRI. ADDR2. IC;
          LOAD IC DEMUX I:=ADDR I . ADDR2. IC;
          LOAD IC DEMUX II:=ADDRII. ADDR1. IC;
     END;

DB:

IF (Q' AND S' AND T') TRUE THEN
     BEGIN
          S1=1;S2=0;LOAD  1:=0; LOAD ADD 1:=0000;
          S4:=0;LOAD IC DEMUX I:=0000;
          LOAD IC DEMUX II:=0000;
     END;

IF ( Q' AND S' AND T )TRUE THEN
     BEGIN
          S1=1;S2=0;S4=0;
          LOAD 1:=1;LOAD ADD1:= ADDRII. ADDR2. IC;
          LOAD IC DEMUX I:=0000;
          LOAD IC DEMUX II:=ADDR II. ADDR2. IC;
     END;
```

```
IF ( Q' AND S AND T') TRUE THEN

     BEGIN

          S1=1;S2=0;S4=0;

          LOAD 1:=1;LOAD ADD1=ADDRII. ADDR1. IC;

          LOAD IC DEMUX I:=0000;

          LOAD IC DEMUX II:=ADDR II. ADDR1. IC;

     END;

IF ( Q' AND S AND T ) TRUE THEN

     BEGIN

          S1=0;S2=1;S4=1;

          LOAD 21:=LOAD 22:=1;

          LOAD ADD21:=ADDRII. ADDR1. IC;

          LOAD ADD22:=ADDRII. ADDR2.IC;

          LOAD IC DEMUX I:=ADDRII. ADDR1. IC;

          LOAD IC DEMUX II:=ADDRII. ADDR2.IC;

     END;

IF ( Q AND S' AND T') TRUE THEN

     BEGIN

          S1=1;S2=0;S4:=0;

          LOAD 1:=1;LOAD ADD1:=ADDRI. ADDR2.IC;

          LOAD IC DEMUX I:=ADDRI. ADDR2.IC;

          LOAD IC DEMUX II:=0000;

     END;
```

```
IF (Q AND S' AND T)TRUE THEN

     BEGIN

          S1=0;S2:=1;S4=0;

          LOAD 21:=LOAD 22:=1;

          LOAD ADD21:=ADDRII. ADDR2. IC;

          LOAD ADD22:=ADDR I. ADDR2. IC;

          LOAD IC DEMUX I:=ADDRI. ADDR2. IC;

          LOAD IC DEMUX II:= ADDRII. ADDR2. IC;

     END;

IF(Q AND S AND T')TRUE THEN

     BEGIN

          S1=0;S2=1;S4=0;

          LOAD 21=LOAD22:=1;

          LOAD ADD21:=ADDR II. ADDR1.IC;

          LOAD ADD22:=ADDRI. ADDR2.IC;

          LOAD IC DEMUX I:=ADDRI.ADDR2. IC;

          LOAD IC DEMUX II:=ADDRII. ADDR1. IC;

     END;


IF (Q AND S AND T )TRUE THEN

     BEGIN
```

```
          UPDATE QUEUE OVF:=1;
     END;


DC:

IF (Q' AND S') TRUE THEN

     BEGIN

          S1:=1;S2:=0;LOAD 1:=0;
          LOAD ADD1:=0000;
          LOAD IC DEMUX I:=-0000;
          LOAD IC DEMUX II:=0000;
     END;

IF (Q' AND S )TRUE THEN

     BEGIN

          S1=1;S2=0;LOAD 1:=1;
          LOAD ADD 1:=ADDR II. ADDR1. IC;
          LOAD IC DEMUX I:=0000;
          LOAD IC DEMUX II:= ADDR II. ADDR1. IC;

     END;

IF (Q AND S') TRUE THEN

     BEGIN

          S1=1;S2=0;S4=0;
          LOAD 1:=1;LOAD ADD1:= ADDRI.ADDR2. IC;
```

```
        LOAD IC DEMUX I:=ADDRI. ADDR2. IC;
        LOAD IC DEMUX II:=0000;
    END;

IF( Q AND S ) TRUE THEN
     BEGIN

        S1=0;S2=1;S4=0;
        LOAD 21:=LOAD 22:=1;
        LOAD ADD21:=ADDRII. ADDR1. IC;
        LOAD ADD 22:= ADDRI. ADDR2.IC;
        LOAD IC DEMUX I:= ADDRI . ADDR2. IC;
        LOAD IC DEMUX II:= ADDRII. ADDR1. IC;

    END;

DD:

IF (Q' AND T') TRUE THEN

    BEGIN

        S1=1;S2=0;S4=0;
        LOAD 1:=0;LOAD ADD1:=0000;
        LOAD IC DEMUX I:=0000;
        LOAD IC DEMUX II:=0000;

    END;

IF (Q' AND T )TRUE THEN
```

```
    BEGIN

        S1=1;S2=0; S4=0;
        LOAD 1:=1; LOAD ADD1:=ADDRII. ADDR2. IC;
        LOAD IC DEMUX I:=0000;
        LOAD IC DEMUX II:=ADDRII. ADDR2.IC;

    END;

IF (Q AND T') TRUE THEN

    BEGIN

        S1=1;S2=0;S4=0;
        LOAD 1:=1;
        LOAD ADD1:=ADDRI. ADDR2. IC;
        LOAD IC DEMUX I:=ADDRI. ADDR2.IC;
        LOAD IC DEMUX II:= 0000;
    END;

IF( Q AND T )TRUE THEN

    BEGIN

        S1=0;S2=0;S4=0;
        LOAD 21:=LOAD 22:=1;
        LOAD ADD21:=ADDRII. ADDR2.IC;
        LOAD ADD22:=ADDRI. ADDR2. IC;
        LOAD IC DEMUX I:=ADDRI. ADDR2. IC;
        LOAD IC DEMUX II:=ADDRII. ADDR2. IC;
```

```
      END;

END CASE
```
